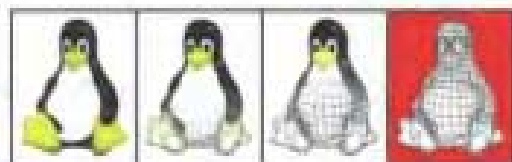


书中有 40 多个完整的、带有注释的脚本，这是学习 Python 更好的方法

Python

编程指南

Programming with Python



[美] Tim Altom, Mitch Chapman 著
云舟工作室 译



中国水利水电出版社
www.waterpub.com.cn

快来，立刻运行Python!

用这些最小的解释和最长的代码来深入探索Python! 本书包含了许多经过测试的脚本，包括数据库的访问、GUI操作、聊天服务器、各种游戏、文件操作以及更多来自世界各地的Python程序员的脚本。用这些脚本学习Python语言，并作为自己编写脚本的起点。

更快地学习

- ▶ 包括Python语句和数据结构方面的信息
- ▶ 把握标准Python模块的特征，解释如何更有效地运用它们
- ▶ 包含一个Python的FAQ，以便帮你迅速地解决问题

更多地学习

- ▶ 包含40多个经过测试的脚本，包括查询搜索引擎、执行统计分析，以及更多……
- ▶ 介绍有经验的Python程序员编写脚本时如何考虑问题
- ▶ 包含如何使用Python开发Internet应用程序的指导

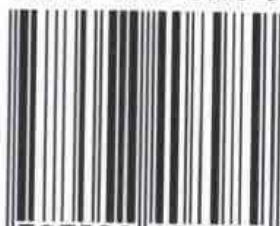


CD-ROM 内容

- ◆ Python 源代码
- ◆ Microsoft Windows 下编译过的Python
- ◆ Grail，一个完全用Python编写的浏览器
- ◆ 专用的Python模块
- ◆ 本书中的脚本



ISBN 7-5084-0898-5



9 787508 408989 >



北京万水电子信息有限公司

Beijing Wanshui Electronics Information Co., Ltd.

地址：北京市西直门外榆树馆一巷永康商务写字楼

邮编：100044

电话：(010)6835.9286, 6835.9167

传真：(010)6835.9284

E-mail: mchannel@public3.bta.net.cn

ISBN 7-5084-0898-5/ TP·341

定价：40.00元(含1CD)

Python 编程指南

[美] Tim Altom, Mitch Chapman 著

云舟工作室 译

中国水利水电出版社

内 容 提 要

本书是一本全面介绍 Python 语言的书籍。作者首先简明扼要地介绍了 Python 的语句、模块、数据类型、函数和模块等基础知识，然后通过大量的示例程序，详细介绍了 Python 在 Tkinter 脚本、数据库、数学/科学函数、服务器、字符串和其他数据类型、系统操作、游戏和人工智能方面的应用。并且作者将 Python 的 FAQ 放在本书的附录中，方便读者查阅。

本书适合 Python 初学者，对于有一定 C 语言基础的人，学习本书将更加容易。

Authorized translation from English Language Edition published by Prima communications, Inc. Original copyright © 1999 by Prima Publishing, Programming with Python. Translation by China WaterPower Press, 2001.

北京市版权局著作权合同登记号：图字 01-2001-2163 号

图书在版编目 (CIP) 数据

Python 编程指南/(美)阿尔托马 (Altom, T.), (美)查普曼 (Chapman, M.)
著; 云舟工作室译. —北京: 中国水利水电出版社, 2001

ISBN 7-5084-0898-5

I. P… II. ①阿…②查…③云… III. Python 语言—程序设计 IV.TP312

中国版本图书馆 CIP 数据核字 (2001) 第 084632 号

书 名	Python 编程指南
作 者	[美]Tim Altom, Mitch Chapman 著
译 者	云舟工作室 译
出版、发行	中国水利水电出版社 (北京市三里河路 6 号 100044) 网址: www.waterpub.com.cn E-mail: mchannel@public3.bta.net.cn (万水) salc@waterpub.com.cn 电话: (010) 68359286 (万水)、63202266 (总机)、68331835 (发行部)
经 售	全国各地新华书店
排 版	北京万水电子信息有限公司
印 刷	北京蓝空印刷
规 格	787×1092 毫米 16 开本 20.75 印张 452 千字
版 次	2002 年 1 月第一版 2002 年 1 月北京第一次印刷
印 数	0001—4000 册
定 价	40.00 元 (含 1CD)

凡购买我社图书，如有缺页、倒页、脱页的，本社发行部负责调换

版权所有·侵权必究

译 者 序

Python 是一种脚本语言。脚本语言是类似 DOS 批处理、UNIX shell 程序的语言。脚本语言不需要每次编译再执行，并且在执行中可以很容易地访问正在运行的程序，甚至可以动态地修改正在运行的程序，适用于快速地开发以及完成一些简单的任务。在这样的情况下，Python 可能正好适合你的需要。

Python 是一门解释性的、面向对象的、动态语义特征的高层语言。Python 的简单而易于阅读的语法强调了可读性，因此降低了程序维护的费用。Python 支持模块和包，并鼓励程序模块化和代码重用。Python 的解释器和标准扩展库的源代码和二进制格式在各个主要平台上都可以免费得到，而且可以免费分发。

由于 Python 的诸多优点，所以其是学习脚本语言者的首选。本书由浅入深地举例说明了 Python 语言的使用，并提供了许多有关 Python 的网站的信息，可以从这些网站得到更多有用的资料。本书附带的光盘提供了书中的示例脚本，以帮助读者更好地学习 Python 语言。

全书分为两大部分，第一部分为第 1 章至第 5 章，介绍了 Python 的基础知识，如 Python 的语句、模块、数据类型、函数和模块等，第二部分为第 6 章至第 12 章，通过列举大量的示例程序，详细介绍了 Python 在 Tkinter 脚本、数据库、数学/科学函数、服务器、字符串和其他数据类型、系统操作、游戏和人工智能方面的应用。作者将 Python 的 FAQ 放在本书的附录中，方便读者查阅。

本书的最大特点在于“脚本”，提供了 40 多个大小不等的示例程序。学习一种语言的初学者最需要的往往是例子。有了例子的帮助，我们可以更快地学习语言，甚至可以从网页或光盘中收集示例代码满足自己的需要。

本书由云舟工作室编译，李国、杨水超、杨作梅、闫娅男、刘洋等同志承担了主要的编译工作。由于水平有限，书中还有很多不当之处，恳请广大读者批评指正。

译者

2001 年 9 月

作者简介

Tim Altom

在 Tim Altom 成年之后的前 15 年里，他做过电子技师和工业程序员。对于他来说能够文理兼顾是必须的，所以他寻求并最终在 Indiana 大学取得了英语/新闻学的学位。最近几年在 Indianapolis，他作过技术通信员、项目经理、撰稿人以及 Simply Written 公司的副总裁和首席技术顾问。他精通各种编程语言，几乎无所不通。他用过的编程语言有 Visual Basic for Applications、汇编、C、JavaScript、Java、Boolean、Basic、Lisp 和现在的 Python。

Tim 是技术通信协会的高级成员和 Hoosier 分会的前任会长。他在 Simply Written 公司关于文档编制方法的许多会议上发表过演讲，并且写过多篇专业的文章，题目从使用“big”一词的历史到为诵读有障碍者设计文件等等。他是 Prima 出版社的《Hands On HTML》一书和其他书籍的合作者。

他已婚，有两个女儿和一个继子，生活在 Indiana 州 Indianapolis 的东北面 Lawrence 的一个舒适、独立的社区中。Lawrence 是 Fort Benjamin Harrison（一个 Base Closure 运动的受害者）的新拥有者。Simply Written 公司很乐意地制定了“再利用”计划，一定程度上说服联邦政府将 Fort Harrison 卖给社区，目前那里已成为 Lawrence 社区生活的中心。

Mitch Chapman

Mitch Chapman 从 1980 年父母给了他一台 TI-55 计算器之后，就一直在编程。他在 Ohio 州 Dayton 的 Wright State 大学获得了计算机科学的学位，然后开始工作。他曾在 Wright-Patterson AFB 维护空气动力学建模软件，并且后来帮助进行瑞士 JAS-39 Gripen 的自动飞行测试。

Mitch 在 Ohio Electronic Engravers 度过了 6 年时间，他以非凡的努力制造出了世界最先进的出版影印系统。

Mitch 已经用 C、C++、Pascal、汇编语言和 Fortran 语言开发了软件，并且还使用过其他多种语言。1996 年他发现了 Python，到目前为止，这是他最喜爱的。

Mitch 目前受雇于 Bioreason，这是一家位于 Santa Fe, NM 刚刚新建的化学信息公司，Bioreason 致力于高吞吐量药物屏幕数据分析的自动化。他们用 Python 做一切事情，从 AI 到 GUI 应用程序。

Mitch 是 IEEE 计算机协会的成员，并很自豪、幸运地成为 Python 软件组织的第六个成员。

致 谢

首先要感谢 Prima 出版社的 Kim Spilker, 他对 Linux/GNU 书籍有极好的洞察力; 感谢 Kevin Harreld, 他作为编辑做了这么多工作。

另外对本书做出贡献的还有:

Mitch Chapman

Andrew Markebo

Sam Rushing

Joe Strout

Michel Vanaken

还有那些宁愿保持匿名的人, 感谢你们中的每一位。

特别要感谢 Mitch Chapman, 他和我在一起深入地研究和测试脚本。祝他的队伍日益壮大。

简介

我不是每天都能写关于一些非常有趣的东西的书。的确，技术本身通常是有趣的，常常使人着迷，偶尔甚至有很高的报酬，但是有趣常常不是等式的一部分。

然而，Python 很有趣，真的很有趣！它强大、灵活、简单，尤其是非常好用。

我是一个交流者，我非常擅长于此，靠它足以维持生活；我还是一个爱开玩笑的人，因此我写、说、讲一些技术问题才是最自然的。但是，许多技术好得使人着迷，差得令人讨厌。多数软件在其界面弹出到表面时没有正确地发出应有的光芒，这么多我在文档里称之为封闭系统的东西，被编译者藏起来。当你掀起顶盖要修车的时候，风挡玻璃刮水器控制装置的资料该是多好的东西啊？

Python 就不同，它是开放的。可以得到源代码，并且事实上它就在本书所附的光盘里。它很简单，用它写代码的人们颇能获益，并有一个很好的共同的幽默感觉。

我偶然发现 Python 是在几年前我装载了第一个 Linux 发行版的时候。在 Simply Written 公司，我们大多在 Microsoft Windows 下工作，这是一个错误（这里我要承认它，然后丢下它不管）。我们的客户大量运行 Windows，因此我们也这么做了。一到 Adobe 公司推出 Linux 平台的 FrameMaker 时，我们才立刻认真地讨论大规模的转化。

在很少的脚本语言的世界里，Python 还是一个很不成熟的选手，这是我也不得不承认的事实。Perl 是第一个而且仍是大多数 UNIX/Linux 用户所选择的脚本语言。与 Perl 相比，Python 更适合 Linux/GNU 来挑战 WenTel（对微软和英特尔结盟分享商业利益的戏称——译者注）这个怪物。Python 比 Perl 更灵巧，更多地面向对象，更容易学。但是为数众多的用户已经学过 Perl 了，并且没有学过 Python。许多用户甚至没有见过 Python 或使用过它。然而，对那些日益被最后期限和迅速成长的需求折磨的管理员和 Webmaster 来说，Python 无疑是一个天赐之物。Python 可以减轻这些压力，需要它的人可以先快速地学习和应用 Python 的初步内容，同时他们可以逐渐地深入到 Python 的细节中去。

本书就是想帮助这些可怜的人。它有别于 Python 以前的书籍，因为它没有用任何渐进指南的方式来教你这门语言。如果你需要，其他几本好书可以做到这一点。

相反，本书介绍了仅有的 Python 骨架：它的语句、模块、数据类型等等。如果你已经非常熟悉 C 语言、数据库的术语和其他管理员类型的技术，你会觉得 Python 的大部分都比较熟悉。

然后，本书提供许多脚本。有些很短，其他的则很长、很复杂。以我的经验，一个语言的初学者最强烈需要的是例子。在可以下工夫研究和尝试的丰富的示例代码的支持下，我们大多数都可以更快地学习语言。你甚至可以从网页或光盘中收集示例代码满足自己的需要。

简 介

我不是每天都能写关于一些非常有趣的东西的书。的确，技术本身通常是有趣的，常常使人着迷，偶尔甚至有很高的报酬，但是有趣常常不是等式的一部分。

然而，Python 很有趣，真的很有趣！它强大、灵活、简单，尤其是非常好用。

我是一个交流者，我非常擅长于此，靠它足以维持生活；我还是一个爱开玩笑的人，因此我写、说、讲一些技术问题才是最自然的。但是，许多技术好得使人着迷，差得令人讨厌。多数软件在其界面弹出到表面时没有正确地发出应有的光芒，这么多我在文档里称之为封闭系统的东西，被编译器藏起来。当你掀起顶盖要修车的时候，风挡玻璃刮水器控制装置的资料该是多好的东西啊？

Python 就不同，它是开放的。可以得到源代码，并且事实上它就在本书所附的光盘里。它很简单，用它写代码的人们颇能获益，并有一个很好的共同的幽默感觉。

我偶然发现 Python 是在几年前我装载了第一个 Linux 发行版的时候。在 Simply Written 公司，我们大多在 Microsoft Windows 下工作，这是一个错误（这里我要承认它，然后丢下它不管）。我们的客户大量运行 Windows，因此我们也这么做了。一到 Adobe 公司推出 Linux 平台的 FrameMaker 时，我们才立刻认真地讨论大规模的转化。

在很少的脚本语言的世界里，Python 还是一个很不成熟的选手，这是我也不得不承认的事实。Perl 是第一个而且仍是大多数 UNIX/Linux 用户所选择的脚本语言。与 Perl 相比，Python 更适合 Linux/GNU 来挑战 WenTel（对微软和英特尔结盟分享商业利益的戏称——译者注）这个大怪物。Python 比 Perl 更灵巧，更多地面向对象，更容易学。但是为数众多的用户已经学过 Perl 了，并且没有学过 Python。许多用户甚至没有见过 Python 或使用过它。然而，对那些日益被最后期限和迅速成长的需求折磨的管理员和 Webmaster 来说，Python 无疑是一个天赐之物。Python 可以减轻这些压力，需要它的人可以先快速地学习和应用 Python 的初步内容，同时他们可以逐渐地深入到 Python 的细节中去。

本书就是想帮助这些可怜的人。它有别于 Python 以前的书籍，因为它没有用任何渐进指南的方式来教你这门语言。如果你需要，其他几本好书可以做到这一点。

相反，本书介绍了仅有的 Python 骨架：它的语句、模块、数据类型等等。如果你已经非常熟悉 C 语言、数据库的术语和其他管理员类型的技术，你会觉得 Python 的大部分都比较熟悉。

然后，本书提供许多脚本。有些很短，其他的则很长、很复杂。以我的经验，一个语言的初学者最强烈需要的是例子。在可以下工夫研究和尝试的丰富的示例代码的支持下，我们大多数都可以更快地学习语言。你甚至可以从网页或光盘中收集示例代码满足自己的需要。

本书的编排

第 1 章至第 4 章是介绍性的。你将读到关于 Python 语句的内容，看到 Python 模块的列表，以及关于 Python 如何组合在一起的若干说明。然而别指望有深入的论述，我要为代码节省篇幅。

我通常使用和测试脚本的 Python 版本是 1.5.2。其他版本在你读本书的时候可能已经出来了，因此请到 www.python.org 上查看更新的信息。许多投稿的脚本是用旧版本写成的，某些情况下需要提供更新的版本。

其余的章节就是例子代码了。这涉及了简单到字符串处理之类，复杂到只有高级用户才能理解的代码。

每个例子脚本都伴有代码的分析和解释。我分解一些脚本的操作，以便你能看到操作的细节。深入内部的工作可以使你快速进入到 Python 中。

还有一个附录，是关于 Python 的一个 FAQ。你可以访问 Python 的 Web 站点 (www.python.org) 来获取最新的 FAQ。Python.org 的 FAQ 真是杰作，应受到赞扬。所有的 FAQ 都有如此好的组织、如此全面、如此广博。你几乎可以通过 FAQ 来学习 Python。它在所附光盘上也有 HTML 格式的文件。

更多的资料

Python 的交换中心在 python.org，下载、文档、SIG——这里应有尽有。

能提供每日帮助的主要来源在 comp.lang.python。在发表任何东西之前，先查看一下 FAQ。它在本书的光盘中，在附录中，在 python.org 上都有。列表的参与者当它们要写一个“查看 FAQ”的帖子时容易生气。正如两个 Monty Python 的人讲的非常著名的幽默一样：

“我已经告诉过你一次了！”

“不，你没有啊？”

“我肯定讲过了？”

脚本的来源

你会注意到大多数的脚本都不是我的。本书的大多数脚本都是别人贡献出来的。其中一些是它们的作者正在使用的，其他则是一些写出来张贴给别人用于取乐的很有趣的程序。

我从 Python 程序员那里要脚本是因为，无论我写多少，我的代码总是我自己的风格。其他用 Python 写的代码则更好，因此我认为，拥有来自众多有经验的程序员的各种代码会更好地帮助你。这些脚本是公用领域的，请根据你的意愿使用它们。然而，大多数作者都希望你在将代码用于商业目的的时候让他知道。在得到许可的地方，我会标明程序员的名字和组织。你也许想联系他们，说声谢谢就可以了。

Python 是自由/开放源码世界的一部分。这意味着你可以根据自己的想法来获取、装载、

运行和修补 Python。这种方法与商业软件通用的做法形成鲜明的对照，商业软件给你一个盒子，要它为它付钱，然后努力去用好它。许多 Python 程序员这样喜欢 Python 是因为它不是私有的。Python 的工具很小，可以像一个产品一样扩展。Python 程序员常常像 Python 本身那样开放和可用。他们通常慷慨地提供建议和帮助，并且在这里投稿的程序员毫无例外地应该受到赞誉。

虽然脚本的作者们很有帮助，但他们并不愚蠢。他们对你拿代码所做的任何事情都不负责任。在机器上尝试代码和排除错误可都是你自己的工作。

我编辑过某些脚本使它更适合我的目的。在这种情况下，我会先展示原始的代码，然后是我的改动。一般我会尝试以简单的例子来开始脚本的每个段落，然后渐渐深入，以给你一个最好的切入点。

Python 和平台

Python 适合每一种适当流行的平台，包括 Amiga。然而，这些平台没有相应对等地“Python 化”。本书中我使用一个 WinTel 的 Python 发行版和一个 Linux 上的版本。它们是不同的。在两个平台上有些脚本不能同样地运行。例如 Shaney.py，在 WinTel 上就不会停止运行，它持续地输出文本；而在 Linux 的机器上很正常。

因此，应该要小心对待仅仅拷贝一个脚本并指望它运行良好的想法。你在自己运行的时候可能会遇到这样或那样的问题。很基础的 Python 脚本应该各处都能运行，但是当你留下尽量简明的代码并做更多的尝试，脚本的可靠性会有所改变。

总之，我宁愿在 Linux 机器上写脚本。我的发行版本是 Caldera 的，我大多在 Xemacs 20.4 上写代码。Python 模式使得它对缩进的跟踪更容易。你可以在 Xemacs 中很好地测试脚本。但是谨慎对待在使用 Tkinter 的 Xemacs 中测试脚本，因为依我的经验，如果尝试在 Xemacs 之外运行，Xemacs 会变慢甚至停止。啊，还好吗？

本书中的脚本都是在 Linux/GNU 上写成并特别针对 Linux/GNU 的，在其他地方这些脚本也许能运行得很好，也许不能。

领略一下 Python

好，如果你能跟得上我这个速度，大概你已经渴望看一些代码了。这里有一个我编辑过的由 Joe Strout 提交的脚本。

```
#!/usr/bin/python
# dog.py
### 获得初始年龄
def doggie():
    try:
        age = input("Enter your age (in human years): ")
```

```

    print # print a blank line
### 进行一些范围检查，然后打印结果
    if age < 0:
        print "Negative age?!? I don't think so."
    elif age < 3 or age > 110:
        print "Frankly, I don't believe you."
    else:
        print "That's", age*7, "in dog years."
### 暂停，等待键入Return键（所以窗口不出现）
    raw_input('press Return')
except SyntaxError:
    print "Why didn't you type anything? Try again."

if __name__=="__main__":
    doggie()

```

脚本中#号后面的任何东西都被当作注释。在上面的脚本中，另外两个#号只是为了强调；只有第一个起作用。

注意缩进的使用。Python 并不使用圆括号“()”或花括号“{}”来包括代码。它只用缩进来表示。def 表示一个函数的开始。

if、while、for 和 def 语句结尾带有一个冒号，提示解释器向下寻找和运行代码块。if 语句后还有 else 和 else if（缩写为 elif）。

这个脚本还有其他各种形式，但你学到了它的思想。你可能已经猜想出它的操作而不需要了解更多 Python 的知识。几分钟的学习，你已经能独立编写基于这个例子的脚本。所以 Python 非常简单，不是吗？

在解释器中运行这个脚本也不困难。在命令行或终端窗口的模式下键入 python，启动解释器。如果 Python 正确地安装了，就会得到如下所示的 Python 提示符，像三个尖括号：

```
>>>
```

确认脚本是 Python 可以访问到的。最简单的办法是将 dog.py 文件复制到 Python 库目录下。在我的 Linux 机器上，该是 usr/bin/lib/python1.5/。

键入如下的命令，运行脚本：

```
python dog.py
```

脚本加载然后运行。在提示符状态下重复上面的命令，就可以再运行一次。

也可以将脚本输入到 Python 会话中，如下：

```
>>>import dog
```

将回到 Python 提示符状态。dog.py 现在已经在 Python 的名字空间里了。要让它再做 doggie 的小把戏，键入：

```
>>>dog.dogie()
```

这使得解释器运行 dog.py 里的 doggie()函数。在屏幕上将看到：

```
Enter your age (in human years):
```

接下来键入你的年龄。不要试图用小于 3 或大于 110 的数来干扰脚本，它知道这一点。键入 12 然后按 Enter 键，你将得到：

```
That's 84 in dog years.
```

```
Press Return.
```

按下 Enter 键回到提示符状态。

如果还想运行这个小程序，只要再次键入 dog.doggie()然后按 Enter 键即可。

这个例子只触及了 Python 的一些皮毛，但它描绘了最基本的原理。

准备好学习更多知识了吗？来吧！

目 录

译者序
作者简介
致谢
简介

第一部分 “啊，你得到了什么？”

第 1 章 Python 的介绍	1
1.1 脚本化：现在有些事情完全不同了	2
1.2 如何选择	3
1.3 在 Python 中面向对象编程：在其他东西上面再放些东西	4
1.4 如何使用本书中的脚本	5
第 2 章 语句和内部命令	6
2.1 语句	6
2.2 算术运算	23
2.3 比较运算	25
2.4 数据类型	27
2.4.1 数字	27
2.4.2 序列	28
2.5 特殊类型的方法和操作	29
2.5.1 字符串序列	29
2.5.2 字符串	31
2.5.3 列表	32
2.5.4 字典	34
2.6 文件	37
2.7 异常 (Exception)	39
2.7.1 异常的列表	40
2.7.2 建立自己的异常	44
2.7.3 处理异常	44
第 3 章 模块	46

3.1	Python 服务.....	47
3.2	字符串服务.....	50
3.3	其他服务.....	51
3.4	普通操作系统服务.....	52
3.5	可选的操作系统服务.....	53
3.6	UNIX 特定服务.....	55
3.7	CGI 和 Internet.....	56
3.8	限制运行.....	59
3.9	多媒体.....	59
3.10	加密服务.....	60
3.11	SGI IRIX 特定服务.....	60
3.12	SunOS 特定服务.....	61
3.13	Microsoft Windows 特定服务.....	62
第 4 章	Tkinter	63
4.1	窗口小部件.....	63
4.1.1	Button (按钮) 窗口小部件.....	63
4.1.2	Canvas (画布) 窗口小部件.....	64
4.1.3	Canvas Arc (画布弧) 项目.....	64
4.1.4	Canvas Bitmap (画布位图) 项目.....	64
4.1.5	Canvas Image (画布图像) 项目.....	65
4.1.6	Canvas Line (画布线条) 项目.....	65
4.1.7	Canvas Oval (画布椭圆) 项目.....	66
4.1.8	Canvas Polygon (画布多边形) 项目.....	66
4.1.9	Canvas Rectangle (画布矩形) 项目.....	67
4.1.10	Canvas Text (画布文本) 项目.....	67
4.1.11	Canvas Window (画布窗口) 项目.....	68
4.1.12	Checkbutton 窗口小部件.....	68
4.1.13	Entry 窗口小部件.....	68
4.1.14	Frame (框架) 窗口小部件.....	69
4.1.15	Label (标签) 窗口小部件.....	69
4.1.16	Listbox (列表框) 窗口小部件.....	69
4.1.17	Menu (菜单) 窗口小部件.....	70
4.1.18	Message (消息) 窗口小部件.....	71
4.1.19	Radiobutton (单选按钮) 窗口小部件.....	71
4.1.20	Scale (标尺) 窗口小部件.....	72

4.1.21	Scrollbar (滚动条) 窗口小部件	72
4.1.22	Text (文本) 窗口小部件	72
4.1.23	Toplevel (顶层) 窗口小部件	73

第二部分 脚本

第 5 章	运行 Python	74
5.1	GUI 模式下的主群组	74
5.2	主群组模式	74
5.3	交互模式	75
第 6 章	Tkinter 脚本	78
6.1	Andy 的窗口小部件	78
6.2	Regexer.py	86
6.3	dictEdit.py	99
6.4	engine.py	112
6.4.1	TkSearch.py	112
6.4.2	它怎样工作	118
6.4.3	TkSearchUI	118
6.4.4	它怎样工作	122
第 7 章	数据库	123
7.1	nickname.py	123
7.2	dbase3.py	124
第 8 章	数学/科学函数	134
8.1	donutil.py	134
8.2	julian.py	138
8.3	roman.py	145
8.4	stats.py	148
第 9 章	服务器	153
9.1	basic.py	153
9.2	form.py	154
9.3	helloworld.py	157
9.4	counter.py	158
9.5	sengine.py	163
9.6	engine.py	168
9.7	who-owns.py	175

9.8	server.py	178
第 10 章	字符串和其他数据类型	183
10.1	strplay.py	183
10.2	dog.py	186
10.3	banner.py	187
10.4	ebcdic.py	191
10.5	lc.py	194
10.6	sample.py	195
10.7	xref.py	200
10.8	piglatin.py	203
第 11 章	系统操作和编程	208
11.1	spinner.py	208
11.2	tabfix.py	209
11.3	python2c.py	212
11.4	otp.py	223
11.5	space.py	226
11.6	tree.py	229
第 12 章	游戏程序和人工智能	232
12.1	logic.py	232
12.2	shaney.py	242
12.3	therapist.py	247
12.4	lotto.py	257
12.5	warmer.py	258
12.6	questor.py	260
附录 A	Python 编程和已知故障常见问题解答 (FAQ)	264
A.1	构建 Python 和其他已知故障	264
A.2	用 Python 编程	265
A.3	构建 Python 和其他已知故障	268
A.4	用 Python 编程	275
附录 B	光盘内容	312
B.1	运行 CD	312
B.1.1	Linux	312
B.1.2	Wondows 95/98/NT4	312
B.2	第一许可	313
B.3	第一个用户界面	313

B.3.1	恢复和关闭用户界面	313
B.3.2	使用左边的窗格	313
B.3.3	使用右窗格	313

第一部分 “啊，你得到了什么？”

这一部分是对脚本语言 Python 的介绍，这部分没有 Python 脚本（脚本都在第 2 部分）。在第 1 部分，你将看见 Python 的语句、数据结构、模块和窗口小部件（Widget）怎样被放在一起。并且第 1 章你会读到与 Python 毫无关系的介绍，语言概述以及 Python 的细节。

第 1 章 Python 的介绍

第 2 章 语句和内部命令

第 3 章 模块

第 4 章 Tkinter

第 1 章 Python 的介绍

本章要点：

- 脚本化：现在有些事情完全不同了
- 如何选择
- 在 Python 中面向对象编程：在其他东西上面再放些东西
- 如何使用本书中的脚本

Monty Python（必胜之蟒，来自一个著名电视短剧集《Monty Python's Flying Circus》的名称，该剧很荒诞、反传统——译者注）是 Python 编程语言的命名灵感，虽然这种联系不那么显而易见。除了名字之外，两者再没有任何东西是一样的。然而，这给了 Python 开发者无穷无尽的令人愉快的旁白和狡猾的影射。对 Monty Python 感到好奇的读者可参观 www.stone-dead.asn.au。

Python 语言是 Guido van Rossum 创建的。他仍然是这个语言的领袖和神父，并且是它的最终决策人。Van Rossum（当 van 在句子的前面出现时，应每位 Guido 的要求用大写字母开头）一直在编写语言已经有好多年了。Python 的起源日期回溯到 1989 年，虽然 van Rossum 直到 1991 年才发布了 Python。从那以后，它被稍微重组了一下，但它基本上仍然是他所创建的相同的语言。Van Rossum 的主页是 www.python.org/~guido/。在给他发送 e-mail 之前请阅读整个主页，因为这样做可以在你询问之前也许就可以找到问题的答案。

在 www.python.org 上有大量关于 Python 的信息。这是 Python 的官方网站以及 Python Software Activity 的主页。具有会员资格需要 50 美元，但是如果你不是成员也可以使用这个网站。在这里，你能找到文档、示例脚本和许多其他的 Python 爱好者。

Python 的当前版本是 1.5.2，有版本 2.0 很快成为事实的传言。1.5.2 可以免费从 www.python.org 上下载。它可用于 Linux、Windows 95/98、Windows NT、Amiga、BeOS、QNX、VMS、Windows CE、Sun Solaris、SGI IRIX、DEC UNIX、IBM AIX、HP-UX、SCO、NeXT、BSD、FreeBSD 和 NetBSD。它在大多数 Linux 包中都是标准的。Python 确实是开源软件，也具有容易用的源代码。

如果没有在对象环境中编写过代码，或者以前没有编写过脚本，对 Python 要做一些解释。这里，我不能讲述关于 Python 的每一件事情，但你可以从 www.python.org 下载大量的信息作为本书的补充。随着 Python 的下载，你也可以得到指南、语言参考和其他资料。

1.1 脚本化：现在有些事情完全不同了

如果你以前编写过脚本，就跳过这一节。往前走，我不会不愉快的。脚本是用脚本语言编写的程序。我知道这是一个循环的定义，但是，在脚本和解译语言之间的界限是很不清楚的。

通常，诸如 Perl、Tcl 或 Python 之类的脚本语言一般是为特定的目的编写的，而且用途有限，不像 C 或 Java 这样成熟的语言有更加普遍的用途，但是这些语言更难学习和使用。脚本中总是为特殊目的编写而且始终都不顺次传递。例如，你可以编写一个脚本每天查找一个文件得到特殊的 URL，脚本做完这些工作就会离开。作为应用程序一部分的脚本化语言有时称为“宏”语言，而且它的小程序称为“宏”而不是“脚本”。其实它们是相同的。

当然，Linux 世界用脚本化语言来装载。Shell 程序有它们自己的脚本化性能，并且在 Linux 中通常能得到 awk、Perl、Python、Tcl、甚至 StarBasic 等。另外，在浏览器中还能使用 JavaScript 和 VBScript。

脚本是被解释而不是编译的。解释程序必须运行，从而确定脚本命令的意思然后为系统解释那些命令。相反，编译的程序不用任何干预而准备直接进入系统。例如，C 和 C++ 是被编译的。Perl、Python、Tcl 和其他脚本语言不是编译的。解释的语言比编译的语言占用的时间长，但通常它们容易使用和维护。

脚本通常用于像过滤数据和重构文件这样的辅助性工作。Web 站点的大部分智能包含在脚本中。脚本能询问数据库、执行计算、确认数据、检测权限、快速生成 HTML、简化网络之间的连接、绘图以及其他等等。脚本可以在命令行的模式下以键入其名字的方式运行，也可以被诸如登录之类的事件来调用。脚本甚至能用于像 Tk 或 Tkinter 这样的 GUI（图形用户界面）以产生一个真正的应用程序。

1.2 如何选择

对于这么多可能的脚本化语言，你怎样选择？而且为什么挑选 Python？

一般，Linux 脚本化语言常用这些类型：

- Shell (Shell 程序)
- Application Specific (特殊应用程序)
- General use (一般使用)

Linux BASH Shell 程序有一个特别有用的脚本化语言，可以完成你想让任何语言完成的大多数基本的事情：操作变量、估计条件、输入和输出数据。然而，这个脚本化语言最常用来做简化艰苦的 Shell 程序运行的方法。顺便说一下，如果你想更多地了解 BASH Shell 程序，在 Linux 命令行键入 `man bash`，如果你有几个小时的空闲时间的话。

特定应用的脚本化语言在应用程序本身中进行操作，不要想让脚本在应用程序之外工作。例如，StarOffice 有 StarBasic 作为它的宏语言。

Python 不能直接和前两种类型竞争。它是更加常用的脚本语言。与 shell 脚本不同，Python 脚本通常可以移植到其他操作系统。Python 脚本一般更灵活和高级，很少和系统命令有关。并且 Python 通常不在应用程序中使用，虽然如果你想让它嵌入应用程序中操作 Python 也能正确使用。

Python 和其他语言共享一般使用类别，尤其是 Tcl 和 Perl。Perl 在 Python 之前很早就出现了而且可能是在世界范围的服务器上最常使用的脚本语言。然而，虽然 Perl 很强大并且灵活，但是它也比 Python 难掌握而且不能像 Python 一样本质上面向对象。此外，在你彻底地学会 Perl 语言之前，它的语法几乎是难以理解的。Python 的语法非常直观，使它容易了解和维护。这种简单性对于初学脚本语言者来说也是一个很大的有利因素。

Tcl (全拼 “tickle”) 也是很流行的，虽然它没有 Python 或 Perl 强大。与 Python 不同，Tcl 没有被设计成一般目的的编程语言。尽管如此，Tcl 向导可以做很多值得注意的事情。

Python 非常擅长与其他环境和语言连接。模块经常用 C 语言来编写，但它们不是必须这么做。Python 解释程序作为 API (应用程序接口) 出现在其他的程序中，使它相对容易嵌入。许多程序员使用 Python 脚本作为封装，或用于 CGI (公共网关接口) 脚本。但 Python 不只是其他应用程序的一个好的玩伴；你用 Python 也能写出完整的应用程序。Grail，一个全功能的浏览器，全部是用 Python 写成的。你将会在附带的 CD 中找到它，或者从 grail.cnri.reston.va.us/grail/ (Grail 主页) 免费下载。Grail 是完全开放的，所以你可以按你想要的那样扩展或修改它。与它编译的同胞相比 Grail 要相当慢。在这方面，Grail 能像 Java 小程序一样使用 Python 脚本。

当你第一次查看 Grail 代码时，你也许会很奇怪地看见主程序是如何短。Python 程序通常是很短的，因为它们能使用许多其他程序的服务，正如下面你将看到的一样。

1.3 在 Python 中面向对象编程：在其他东西上面再放些东西

在 Python 程序员看来，凡事都是对象。对象，在编程语法中是部分暴露在表面的代码的自封单元。他们可以再次使用而且是可移植的。面向对象语言不用实际了解任何关于机器是如何工作的事情，就可以容易地延伸和使用其他程序的内部装置。

例如，考虑在它的内部有几种性能的程序。这些性能相互之间无关。它们看起来像工具箱中的工具。你可以称这些性能为“类”、“函数”或“方法”。

例如，它可以有数学函数，用来计算行星轨道、创建不规则体，计算圆周率并描述任何规模的团体聚会所需要的啤酒。

数学运算非常好，但现今你感兴趣的是 beer 函数。如果你正在准备研究生的方向，beer 计算器对于规划啤酒购买做得很精细。你不需要写一个自己的 beer 计算器。相反，用 Python 写一小段代码调用计算器程序的 beer 计算器部分。你不必知道或关心计算器程序也做了其他事情。事实上，使用 Python 可能用两行代码就激活了 beer 计算器。它就是那样简单。

在这个例子中，你的定位程序和计算器程序（从这开始我称它为“模块”）都是对象。你的定位程序不需要了解计算器是怎样工作的。

你只需要用在汽车中使用收音机的同样方法在计算器模块中使用 beer 计算器，不用考虑点烟打火机是怎样工作的，即使它们在同一台电子系统上。

OOP(面向对象编程)还有许多其他方面，当你浏览本书的脚本时，你会看到一些。Python 使和对象工作变得如此简单，以至于你从来不需要了解全部细节和 OOP 怎样工作的隐蔽的秘密。虽然如此，了解一点在 Python 代码中对象调用如何实现是很有帮助的。

Python 的大量动力来自于和 Python 包一起出现的模块的巨大集合。有时你可以照字面意义构建自己的程序，通过首先调用一个模块、然后是另一个，接着又一个直到你做完为止。

下面是一个典型的 Python 程序，使用一个虚构的模块。

```
#!/usr/local/bin/python
import mathmodule #加载数学模块
beer_needed = mathmodule.beerbust(100) #开始运行函数
#将结果返回给变量"beer_needed"
print beer_needed #在屏幕上打印结果
```

这就是它的全部了。其他模块用大致相同的方法使用。第 3 章有模块和它们能做什么的列表。

Guido van Rossum，关于 Python 所有事情的预言家，说过 Python 的巨大的实力之一是它将刮起使用的旋风。通常，他是对的。可是，我补充的是，Python 的另一个巨大实力是它的召唤性的名字。你可以只使用 Perl 或 tickle 完成这些工作，但是 Python 的世界关于引喻和双关语是无穷尽的。

1.4 如何使用本书中的脚本

本书的脚本打算用于两个目的：教育和练习。学习语言最好的方法是用有用的代码做一些事情，然后看看当你在这或那改变某些东西后发生了什么。当你变得更加熟练以后，在你自己的环境构建自己的特殊的脚本时，这些脚本会给你一个脚手架。

每个脚本都伴随着分析和注释。并且每个脚本都有自己的创建者的名字和其他的像可用性或授权这样的信息。

这些脚本基本上针对 Linux/GNU。有一些脚本可以在任何平台上运行。所有的脚本都已经测试过了，但要记住成功的脚本执行通常很大程度上依赖许多其他因素。例如，如果你可用的都是 DBM，那么编写的与 GDBM 数据库接口的脚本可能不会工作。然而，在两种情况下数据库的工作原理是相同的，即使语句有点不同。在第 7 章你会看到更多有关数据库的知识。

有关免费/开放资源世界的非凡事情之一是，使用它的人共享而且一起很好地运行。如果你决定写一些有趣的 Python 脚本，特别是如果你在这里开始使用一些脚本，我力劝你把它们张贴在某处，使其他人也可以使用。

本书的大部分脚本都很短。这里有一个原因。你看，涂上墨水的死树木切片（意指纸张——译者注）制造和发行起来很昂贵；提交到本书的更长的脚本会打印出数十页或更多。把几个长脚本放入单独的一章，这本书就非常厚了。我必须做出选择。我可以真实地印刷几个脚本，确实很长的脚本，或者是几个具有不同风格和不同方法的较短的脚本。我选择了后者。除此之外，按它本身的大小对于理解存在很大障碍。当你有一个运行成百上千的行时，在某一点后你将会错过逻辑路线。最好还是学习比较容易分析的短脚本。

较短而且更多变化的脚本的着重点有一个缺点。为单一、简单的目的编写的 Python 脚本，在思考模式方面与响应大而有野心的需求相当不同。很少有短脚本使用类定义，例如那是 Python 的优点。模块例行公事地使用类定义。当你需要一个类时，通常可以在库中找到它，所以在本书你将会看到很少的定义类的脚本。然而，这有几个脚本定义了类，正是如此，你能看出它是怎样工作的。尽管你会看到许多被输入和使用的模块的例子。聪明地利用 Python 库能缩短工时。

请享受你的 Python 旅途。如果你有愿意捐献给本书未来版本的脚本，或者也许提示给你，欢迎发 e-mail 给我 taltom@simplywritten.com。

第 2 章 语句和内部命令

本章要点:

- 语句
- 算术运算
- 比较运算
- 数据类型
- 特殊类型的方法和运算
- 文件
- 异常

Python 是一种群体动物，因为它们的强大来自于它的无缝使用模块的能力，而不是必须把许许多多的命令集中到自己的语言资料库中。模块能添加无限种附加命令，这些命令像 Python 的内部命令一样容易。和 Python 一起出现了许多这样的模块，另一些可以在 Web 上得到，同时你可以很容易地写自己的模块。但是，Python 有许多不需要外部模块的内部命令。本章概述了这些内部的操作。

在交互方式下从键盘上可以正确地操作大部分内部语句和函数。在这个模式中，你能够经常立即得到返回的数值。例如，在交互方式中输入语句 `chr(100)` 就会立即工作，把数值打印到屏幕上。本章中给出了这些完整的小例子，就好像你在交互方式中键入了它们一样。

另一方面，在实际脚本中，大部分语句通过给变量分配输出来使用。自启动脚本也许可以使用下面的语法，代替让语句在交互方式下正确打印到屏幕上：

```
character=chr(100)
```

2.1 语句

abs(number)

返回一个整数或浮点数的绝对值。

例子:

```
abs(-6)
```

返回

`apply(function,tuple),apply(function,tuple,dictionary)`

在运行时允许分配一个参数序列，而不必预先定义。如果“tuple”参数还不是一个元组，在使用它之前就会被转换。

例子：

3 个参数形式：

```
apply(do_something, (people), mydictionary)
```

在这个例子中，`do_something` 是一个已定义函数，`(people)` 是一个元组，`mydictionary` 是一个已定义字典。这个字典必须有作为关键字的字符串。

在哪能找到：

第6章：regexer.py

`assert`

检查你为调试目的特意插入的特定调试表达式的真实性（或者，更加有用性，虚假性）。这个语句只有在`__debug__`变量为真时才工作。

例子：

在交互模式下：

```
assert 1>2
```

产生

```
Traceback (innermost last):
```

```
File "<stdin>", line 1, in ?
```

```
AssertionError
```

`break`

中断 `while` 循环的执行并使循环的执行退出。如果当前循环在另一个循环中，`break` 会把执行发送到最近的上一级循环。请参阅 `while`、`continue`。

例子：

```
while x != 0:
```

```
    y = doit(n) # 执行操作的函数
```

```
    if y > 45:
```

```
        break # 如果 y 太大，退出
```

```
    x = x - 1
```

`callable(object)`

检查对象是否是可调用的。如果对象是可调用的，函数返回真；如果对象不是可以

调用的，则返回假。可以调用函数包括函数、类、方法和内置函数。

例子：

```
callable(string) #测试调用变量
```

返回

0

```
callable (callable) #测试调用函数
```

返回

1

chr(x)

返回单个字符串，它是数字 x 的 ASCII 等价物。请参阅 ord(x)。

例子：

```
chr(100) #返回 ASCII 100 的等价值
```

返回

'd'

cmp(x,y)

比较这两个对象 x 和 y，并返回反映等值的整数。cmp()用于比较数字或字符串。如果 x 小于 y 返回的值为负数，如果 x 等于 y 返回零，如果 x 大于 y 返回正数。如果这两个比较的对象是字符串，Python 用标准的字母表顺序比较它们。

例子：

```
cmp(2,3)
```

返回

-1

```
cmp("Graham", "Cleese")
```

4

coerce(x,y)

返回一个由两个被转换为普通类型的数值参数组成的元组。Python 与数字类型不同，它不允许数字操作。它们用 coerce(x,y)转换为普通类型。Python 喜欢长整数要胜于简单整数，并且喜欢浮点数要胜于长整数和简单整数。如果你在别处需要原始的数据类型，要记住你必须向回转换一个或两个参数。

例子：

```
coerce(12, 12L)
```

返回

```
(12L, 12L)
```

compile(string,filename,kind)

把字符串编译进代码对象，从文件名中读取（如果使用）。在提交到解释程序以前，Python 代码被编译成字节码。编译语句预编译一个对象，从而节省运行时编译的时间。产生的这种对象由编译的代码的种类决定：`evals`、`execs` 或 `single`。`execs` 是包含一系列表达式的对象，不返回数值。`evals` 是计算单个表达式的对象。它们能返回数值，但不能将数值赋给变量。`single` 是单个交互语句，它可以是也可以不是一个表达式语句。参阅 `eval()`、`exec()`。

例子：

```
mycharacter=compile("chr(100)","","eval")
```

#将字符串编译成 eval()

调用 mycharacter

```
eval(mycharacter)
```

得到'd'

continue

在 `while` 循环的反复操作中终止后来操作的运行，并向后发送执行到下一个反复。

例子：

```
while x != 0
```

```
    a = a+1 #数值加 1
```

```
    if a<10:#如果还不到 10
```

```
        continue #执行下一个循环
```

del

从列表中删除一个变量或项目。参阅 `lists,variables`。也从名字空间中的类、实例或模块中静态地删除一个属性。

例子：

从列表中删除一个项目：

```
del mylist[0] #从列表中删除第一个项目
```

删除一个变量：

```
del mylist #同时删除变量内容和名字
```

从对象中删除一个属性：

```
del mymodule.yourattribute
```

在哪能找到：

第 6 章：dictEdit.py

delattr(object,name)

在名字空间中动态地删除类、实例或模块的属性。参阅 del、getattr()和 setattr()。

例子:

```
delattr(mymodule, "haircolor")#动态地删除
# mymodule 的属性"haircolor"
```

dir()

在本地符号表中返回名字。dir()返回模块、类和实例的列表。dir(name)返回（名字）属性的列表。这个函数必定是不完整的，例如，它不能从类实例中返回方法。

例子:

```
import mymodule
dir()
产生
['_builtins_', '_doc_', '_name_', 'mymodule']
dir(mymodule)
得到
['yourattribute']
```

elif

“else if”语句的简写。当 if 语句有多个可能的输出时使用。注意在 elif 语句后面一定要跟有冒号，正像用 if 语句一样。elif 后能随意跟一个 else 语句。参阅 if、else。

例子:

```
if myvariable<10:
    doit()
elif myvariable>10:
    dosomethingelse()
```

else

给 if 语句一个可供选择的动作。

例子:

```
if thisvariable != purchaseprice:
    walkout()
else:
    punch()
```

`eval(expression[,globals[,locals]])`

在运行时为单语句对象或其他表达式调用 Python 解释程序。参阅 `exec()`、`compile`。
`expression` 参数是包含 Python 表达式的字符串或用 `compile()` 语句产生的编译代码对象，使用“eval”作为它的第三个参数。另两个参数是可选的字典。

如果提供了 `globals` 参数，那么当计算表达式时它将作为全局名字空间使用。如果也提供了 `locals` 参数，它将作为局部名字空间使用；否则 `globals` 参数将同时作为全局和局部名字空间使用。如果两个字典都没有提供，表达式将使用执行 `eval()` 的名字空间计算。

例子：

计算一个预编译表达式：

```
mycharacter=compile("chr(100)", "", "eval")#创建 mycharacter  
eval(mycharacter)
```

结果为

```
'd'
```

也可以计算一个表达式：

```
myvariable=1  
eval('myvariable+1')
```

得到

```
2
```

`execfile(file[,globals[,locals]])`

读取并执行一个文件但不把它输入名字空间。`globals` 和 `locals` 参数是可选的字典。如果提供 `globals` 参数，当执行文件的内容时它被用作全局名字空间。如果提供 `locals` 参数，当执行文件的内容时它被用作局部名字空间；否则 `globals` 参数同时用于全局和局部名字空间。如果两个字典都没有提供，那么文件的内容在 `execfile()` 被调用的名字空间执行。

例子：

```
execfile('module1.py', mydict)#执行文件  
# 使用字典 mydict 用作全局和局部名字空间  
#"module1.py"
```

`filter(function,sequence)`

如果为了产生已筛选的数据集的语句是基于定义的，排除费解的需要。需要检测过滤器规范和序列。如果函数作为参数给出，`filter()` 返回一系列 `true` 元素。如果使用 `None` 函数，序列检测为“true”并且所有的 `false` 或空元素都不返回。

例子：

设置和检测一列:

```
mylist=[1,2,3,4,5,6,7,8,9]
filter(mydefinition,mylist)#应用过滤定义 mydefinition 用于变量列表 mylist
```

for

构造 for 循环。在 Python 中, for 循环优化重复序列操作。对于构造使用计数器的循环, while 循环比较好。参阅 while、if。

例子:

```
forexample = ("Come in","Look around","Call the cops")
for x in forexample: #迭代序列
    print x #打印序列变量
```

打印的结果

```
Come in
Look around
Call the cops
```

在哪能找到:

第 6 章: Andy 的窗口小部件

float(x)

把字符串或数字转换为浮点类型。参数可以是纯整数或长整数或浮点数字。参阅 int()、long()。

例子:

```
float("-12")#浮点化一个字符串
```

返回

```
-12.0
```

```
float(12)#浮点化一个数字
```

返回

```
12.0
```

getattr(object,name)

在名字空间中动态返回类、实例或模块的属性。参阅 setattr()、delattr()。

例子:

```
getattr(mymodule, "haircolor")#动态地从 mymodule 中获得“haircolor”属性
```

globals()

返回当前全局字符表的字典。

例子:

```
globals()#获得全局名字空间的字典
```

hasattr(object,name)

检查一个给定的对象是否有一个状态的属性。如果为真返回 1，如果为假返回 0。

例子:

```
import math #引入math模块
hasattr(math, 'sin')#询问math模块是否具有“sin”属性
1
```

hash(object)

如果对象是可散列的则返回散列值。散列值是一个整数，并且在字典查找过程中用于比较字典关键字。

例子:

```
hash("cheese shop")
返回
666332922
```

hex(x)

把整数转换为十六进制字符串。

例子:

```
hex(12)
得到
'0xc'
```

id(object)

返回对象的 ID，它实际上是地址。ID 是整数，它对于对象的生存期是唯一的。

例子:

```
id(math)#获取math模块的id
对于这个语句，给出的是:
7870496
```

if

构造一个 if（如果）表达式。if 表达式检查条件并作用于它所找到的条件。参阅 while、for、else、elif。

例子：

```
if thisvariable < thatvariable: #比较两个变量
    jumpup() #执行函数 jumpup
else:
    jumpdown() #执行函数 jumpdown
```

在哪能找到：

几乎任意脚本中都能找到。

import <module>

from <module> import <module methods>

from <module> import*

import <module>把模块的名字输入名字空间并对它进行初始化。当使用这种形式时，在树形中对模块的函数的访问是 module.method()。from <module> import <module methods>不输入模块名字本身，而是只按你列出的名字输入那些模块的函数。然后这些函数被直接调用。from <module> import*在模块中输入所有的函数，但是没有模块名字本身。

例子：

```
import math#根据名字导入模块 math
from math import sin #只从 math 中导入 sin 函数
from math import * #导入全部 math 函数
```

在哪能找到：

在使用模块的任意脚本中。

input(prompt)

等价于 eval(raw_input(prompt))。显示提示符并允许计算表达式。

例子：

```
input("Want me to add those for ya?")
```

得到

```
Want me to add those for ya?
```

然后你可以键入像下面这样的东西：

```
Want me to add those for ya? 4+5
```

并且 Python 返回

在哪能找到：

第10章：dog.py。

int(x)

把字符串或数字转换为纯整数。参阅 long ()、float()。

例子：

```
y=int("34") #生成变量 y 并赋以一整数结果，以得到变量值
34
```

intern(string)

把字符串放入“保留的”字符串的特定表格中，然后返回该字符串。经过 intern 函数处理过的字符串在查找中使用比较快。保留的字符串永远不会被作为无用信息收集。

例子：

```
intern("Doctor, where's my intern?")
Doctor, where's my intern?
```

isinstance(object,class)

如果对象是参数类的实例返回真。如果不是则返回假。

例子：

```
isinstance(myobject, myclass)
1
```

issubclass(class1,class2)

测试这两个类，来查看 class1 是否是 class2 的子类。类是它们本身的子类。

例子：

```
issubclass(myclass1, myclass2)
1
```

len()

返回字符串、列表、元组或字典的长度。有些是变量或直接量。

例子：

```
simplestring = "12345" #向变量指定 5 个元素
print len(simplestring) #计算变量的元素值
打印
5
```

list(sequence)

提供从序列中创建列表的方便的方法。创建的列表与序列的顺序相同。如果序列已经是一个列表，Python 返回该列表的副本。

例子：

```
list("12345")
```

返回

```
['1', '2', '3', '4', '5']
```

locals[]

返回局部字符表的字典。

例子：

```
locals() #获取局部字符表的字典
```

long(x)

把字符串或数字转换为长整数。参阅 int()、float()。

例子：

```
long(2e3)
```

打印

```
2000L
```

map(function,sequence)

在序列中给每个项目应用函数，以产生容纳结果的新列表。

这个函数应该是单个参数。如果函数采用若干参数，那么可以给 map()提供若干序列；第一次函数将从每个列表中用第一个参数调用，然后是第二个，以此类推。

例子：

```
l = [1, 2, 3]
```

```
map(float, l)
```

得到

```
[1.0, 2.0, 3.0]
```

对于用多个参数的函数

```
def expound(stepNumber, step):
```

```
    return stepNumber + ": " + step
```

```
stepNumbers = ["One", "Two", "Three"]
```

```
steps = ["Grasp the holy hand grenade.",
```

```
'Holding it firmly,pull out the pin.',  
 'Count to three.']  
map(expound, stepNumbers, steps)  
得到  
['One: Grasp the holy hand grenade.',  
 'Two: Holding it firmly, pull out the pin.',  
 'Three: Count to three.']
```

在哪能找到:

第11章: python2c.py

max(sequence)

返回序列中最大项目。参阅 min()。

例子:

```
max(7,8,78)
```

产生

78

min(sequence)

返回序列中最小项目。参阅 max()。

例子:

```
min(7,8,78)
```

返回

7

oct(x)

从整数参数中返回八进制字符串。参阅 hex()。

例子:

```
oct(9)
```

返回

'011'

open(name)

open(name,mode)

open(name,mode,bufsize)

从现有文件中创建新文件对象。open(name)打开一个命名文件。open(name,mode)用三

种模式之一打开文件：读、写或附加（“r”、“w”或“a”，用“r+”表示读/写）。
`open(name,mode,bufsize)`允许你为文件指定缓冲器的大小。

例子：

```
open("bruce.txt") #将 bruce.txt 创建为一对象  
open("bruce.txt", "w") #采用写模式打开 bruce.txt
```

在哪能找到：

第 7 章：dbase3.py

ord(x)

返回字符的 ASCII 数值。`ord()`与 `chr()`刚好相反。参阅 `chr()`。

例子：

```
ord('d')  
返回  
100
```

pow(m,n)

pow(m,n,z)

`pow(m,n)`返回 m 的 n 乘幂。`pow(m,n,z)`返回 m 的 n 乘幂，并以模数 z 取模。你也可选用 `pow(m,n)%z` 执行模数，但 `pow(m,n,z)`更有效。

例子：

```
pow(34,5)  
返回  
45435424
```

pass

空语句，什么也不做。当语句需要适当的语法时，填入 `pass`（传送），但什么也不会发生。

例子：

```
while variable1 > variable2  
    pass #暂停，更新变量
```

print

在屏幕上输出编程的文本。

例子：

打印一个项目：

```
print myvariable
```

在同一行上打印若干项目:

```
print name, address, phone
```

在相继的行上打印若干项目:

```
print name, address, phone,
```

range(upper limit)

range(lower limit,upper limit)

range(lower limit,upper limit,step)

在给定范围内作为列表返回所有的整数。**range(upper limit)**从零开始产生简单序列,具有停止点的上限。**range(lower limit,upper limit)**产生带有作为出发点的下限的整数的序列。**range(lower limit,upper limit,step)**产生在下限起始的整数序列,并通过整数步长计算直到上限。参阅 **xrange()**。

例子:

```
range(10)
```

按如下打印:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
range(5,10)
```

按如下打印:

```
[5, 6, 7, 8, 9]
```

```
range(5,10,2)
```

按如下打印:

```
[5, 7, 9]
```

raw_input()

raw_input(prompt)

从标准输入设备得到输入,最常用的是键盘。如果使用 **raw_input(prompt)**会在要求用户输入区的前面打印提示符。键入的输入作为字符串返回。

例子:

```
name=raw_input('Please type somebody's name: ')#将输入保存为变量"name"
```

这条语句产生:

```
Please type somebody's name:
```

当用户键入名字时,它被保存进变量“name”。

在哪能找到:

第7章: **nickname.py**

reduce(function,sequence)

reduce(function,sequence,initialization)

`reduce(function,sequence)`在 `sequence` 参数中逐个对项目应用 `function` 参数，使序列变为单个数值。这个过程对序列中的每个项目进行累积操作。`reduce(function,sequence,initializer)` 添加可选的初始对象，启动序列，并且如果序列为空时则是默认值。

例子：

把数字序列加在一起：

```
x=reduce(lambda a, b: a+b, [1,2,3,5,7,11]) #x 成为 lambda 函数的最终结果
```

现在 `x` 为 29。

这个例子建立了一个代数运算((((1+2)+3)+5)+7)+11)。

把数字序列乘在一起

```
x=reduce(lambda a,b:a*b), [1,2,3,5,7,11]
```

结果为：

66

```
x=reduce(lambda a,b:a*b, [1,2,3], 'bruce')
```

在 `x` 中奇怪地输出：

```
'brucebrucebrucebrucebrucebruce'
```

乘法运算符 (*) 在字符串和数字上的操作不同，但它在字符串和数字上都可以操作。

对于数字来说，它乘上数字。对于字符串重复字符串调用的次数，乘法算子不能作的一件事是把两个字符串相乘。

reload(module)

重新加载并重新初始化先前加载的模块。通常用于模块被修改后重新加载。这个模块必须已经至少加载过一次，或者至少是具有已经加载过的名字的模块。

`reload()`和一些警告以及特殊的考虑事项一起出现。检查你获得的 Python 的正式的 Python 文档。

例子：

```
reload(mymodule)
```

repr(object)

返回一个字符串，它是对象参数的转换。另一个可供选择的方法是使用反引号 ('object')。

例子：

```
repr(123)
```

创建

```
'123'
```

return

在函数定义中用于从函数中返回数值。如果在 `return` 之后没有表达式，那么默认地返回特殊值 `None`。

例子：

```
def example(a,b):  
    total=a+b  
    return total #返回变量 total
```

round(number,places)

返回浮点值，四舍五入位置是参数定义的数字。默认的位置参数是零。

例子：

```
round(1.4567,2)
```

返回

```
1.46
```

setattr(object,name,value)

在命名对象中动态设置属性。‘`setattr(sweetshop,"product","spring surprise")`’和 `sweetshop.product="spring surprise"` 是等价的’。

例子：

```
setattr(test_module, "coolattribute", "albatross") #设置属性"coolattribute"  
#以读取"albatross"
```

str(x)

从 `x` 中返回一个转换字符串。它和 `repr()` 的使用方法一样。

例子：

```
str(6*3) #将 6*3 的结果转换为一个字符串
```

返回

```
'18'
```

tuple(sequence)

把列表序列转换为元组。这项操作把可变的列表制成不可变的元组。通常用于生成字典关键字。

例子:

```
tipple_tuple = ["rum", "merlot", "gin"]#创建一列表
tuple(tipple_tuple)#转换为一元组
tipple_tuple #打印变量的内容
返回
('rum', 'merlot', 'gin')
```

type(object)

返回参数 object 的类型。

例子:

```
import math #导入模块到名字空间
type(math) #检测模块类型
返回
<type 'module'>
```

vars(object)

返回 object 的数值的字典。这个对象必须有_dict_属性。

例子:

```
class demothang #制作一个演示类
    cat="hairball factory"
    dog="loyal companion"
vars(demothang)#获取 demothang 的属性值
和字典一同返回
{'cat': 'hairball factory', 'dog': 'loyal companion', '_doc_': 'none',
 '_module_': '_main_'}
```

while

创建一个只要测试条件存在就继续执行操作运算的 while 循环。

例子:

```
def loop(x,y): #带有 while 循环的函数
while x>y: #测试 x 是否大于 y
print y #打印变量 y
y=y+1
    return y
```

调用这个例子的循环:


```
loop(6,3)
```

将计算函数 `loop(6,3)` 并产生:

```
3
4
5
6
```

xrange(upper limit)

xrange(lower limit,upper limit)

xrange(lower limit,upper limit,step)

在给定范围内作为元组返回整数的 `xrange` 对象。`xrange(upper limit)`生成从零开始的简单序列，并具有参数数值的上限。`xrange(lower limit,upper limit)`生成一个整数序列，并具有作为起始点的下限。`xrange(lower limit,upper limit,step)`产生在下限起始的整数序列，并通过整数的步长计算直到上限。参阅 `range()`。

例子:

```
xrange(10)
```

打印为

```
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
```

```
range(5,10)
```

打印为

```
(5, 6, 7, 8, 9)
```

```
range(5, 10, 2)
```

打印为

```
(5, 7, 9)
```

2.2 算术运算

像其他语言一样，Python 有简单的算术运算。

a+b

把两个数量相加。用于操作字符串和数值。

例子:

```
2+3
```

结果

```
5
```

```
"stark"+"naked"
```

返回

```
'starknaked'
```

a*b

把两个算术量相乘。它用于操作字符串、整数和长整数。如果最终结果可能是长整数，你必须使操作数为长整数。

例子：

```
89898L*999L
```

结果为

```
89808102L
```

a-b

从 a 中减去 b。

例子：

```
17-6
```

得

```
11
```

a/b

用 b 除 a。如果 a 和 b 是整数，则放弃余数。作为提醒，请参见 `modulus`、`next`。正如你所期望的，如果 b 是零，则就引发 `DivideByZero` 异常。

例子：

```
12/5
```

使 Python 返回

```
2
```

a%b

被 b 除的模数。

例子：

```
13%5
```

返回一个模数

```
3
```

-a

获得一个负数（一元减号）。

例子：

```
x=-6 #x=-6
x #打印 x 的值
返回
-6
```

+a

获得一个正数（一元加）。

例子：

```
x= -6 #x=-6
x #打印 x 的值
-6
x=+6 #x=+6
x #打印 x 的值
6
```

divmod(x,y)

在元组中用 y 除 x 并返回作为模数的余数。

例子：

```
divmod(13,5)
得到
(2,3)
```

2.3 比较运算

用于在许多方法中进行比较。在交互模式中，你能直接比较两个对象，它们可以是字符串或数字。例如，如果 x 是 4 而且 y 是 4，那么 `x==y` 会得到一个 1，意味着它为真。如果 x 是“holy”而 y 是“grail”，`x==y` 返回 0。

当从内存中执行脚本时，你能给下面的比较的返回数值分配对象：`a=x==y`。a 变成 1。或者你能在循环中使用比较：`while a<>b`。

x<y

测试 x 比 y 小的事实。

x<=y

测试 x 小于或等于 y 的事实。

x>y

测试 x 比 y 大的事实。

x>=y

测试 x 大于或等于 y 的事实。

x==y

测试 x 等于 y 的事实。

仔细区分分配的单个等号 (=) 和比较的双等号 (==)。Python 可能会捕捉语法错误，它也许会让你花费一些时间尝试指出为什么创建的所有等号不相等。

x◇y

x!=y

测试 x 不等于 y 的事实。这两个表达式中的任何一个都可以用。

is

测试相等的对象。不检查同样名字的对象，但检查同样标识的对象。为假时返回 0，为真时返回 1。

例子：

```
class fishdance: #创建一个 silly 类
    move="slap"
x=fishdance #创建类 fishdance 的一个实例 x
x is fishdance #测试是否相等
1
```

1 表示相等。任何对象的比较都可以为真，包括元组、列表和字典。记住“相等”不意味着“具有同样的元素”。在这个例子中，z 和 y 是相同的对象。

```
y=1,2,3 #创建一个元组
z=y #使 z 等于 y
```

```
z is y
```

```
1
```

在这个例子中，它们不是相同的对象：

```
y=1,2,3
```

```
z=1,2,3
```

```
z is y
```

```
0
```

具有同样的元素，但是不同的对象。

2.4 数据类型

Python 的数据结构都被认为是类型。通过使用 `type()` 可以标识任意数据对象的类型。Python 不同于许多语言，因为它有两种数据结构：可变的和永恒的。正如名字所暗示的一样，可变数据结构能随意改动。你能删除元素、添加元素或清除掉整个结构。永恒数据对象不能遭受这些改动。一旦创建，它们就以它们将要成为的方式存在。

2.4.1 数字

不像某些语言需要你说明想要使用的变量的类型——整型、浮点型，无论是什么——当你输入它时，Python 都会自动为你指出它的类型。

complex

允许可带有复数的操作。任何带有 `j` 或 `J` 的数字都成为一个复数。

例子：

```
89j
```

float

浮点的简称。可惜的是，在 Python 中浮点数字的精确度依赖于你正在使用的平台。

例子：

```
.00000987
```

```
67.26e8
```

int

整数类型。在 Python 中，用于大多数非浮点数字。

例子：

```
6879
```

long

当典型的整数引起溢出时使用。当犯了一个使用太大整数的错误时，Python 会明确地告诉你。在长整数的后面放置一个 L，它就是一个长类型。事实上，它能成为你想让它成为的那么长，只要有内存来容纳它。

例子：

```
95869687364758589747L
```

2.4.2 序列

Python 有 3 个基本的序列数据类型：列表、字符串和元组。用像 `x=[0,1,2,3]` 这样的代码或用像 `x=range(4)` 这样的语句，来明确地创建序列，让 Python 做些艰苦的工作。

list

包含字符串或数字的序列。它是可变的并用方括号环绕。在其他语言中称为数组。

例子：

```
mysequence=[3,4,5]
```

string

字符串是字符的序列。Python 没有字符数据类型；它只是一个长度的字符串。字符串用引号断开，单引号或双引号。参见 Python 手册关于单引号和双引号的规则。

例子：

```
thisstring="It's a fair cop."
```

tuple

元组是永恒的序列，一种自封的列表。元组与列表相比的优越性是，元组是不能轻易改变已知量，从而加强随后的操作运算。元组被集中在圆括号中，另外在它们内部有易变对象。因为在 Python 中元组被如此地频繁使用，尤其是作为字典的关键字使用，构成元组很简单。任何由逗号分开的分组对象都可以成为一个元组。在后面跟有附加物单个逗号的单个对象可以成为一个元组。

例子：

```
a=1,2
```

```
a #打印 a 的值
```

```
b=(1, #创建一“1”对象元组)
```

```
b #打印 b 的值
```

```
a=(1,2) #这仍起作用
```

你能在其他语句中创建元组。例如，创建和使用元组的 `apply()` 的用法如下：

```
def __init__(self, x=5, *args):  
    apply(MyBaseClass.__init__, (self,) + args)  
    self.x = x
```

你只需记住使用结尾的逗号。如果没使用它，就不能得到一个元组。

dictionary

在其他地方称为键控、联合或索引数组。映射可散列化的 Python 对象为其他 Python 对象。关键字是永恒的；数据对象不是永恒的。一组关键字和对象都能被删除。关键字和它的对象在列表中用冒号 (:) 分开。字典允许以跟踪的目的把不同的事物配对。Python 可以非常有效地使用字典。字典包含在花括号中。

例子：

```
x={"cops":"Elliot Ness", "robbers":"Willie Sutton"}
```

2.5 特殊类型的方法和操作

经常有一些等价或类似于语句的情况。Python 做每件事都有多种方法，所以在这里你能找到许多和语句部分交叉的地方。

2.5.1 字符串序列

sequence[index]

在索引点返回对象。索引是首先从零开始计算的整数。

例子：

```
cowboytype="Rhinstone" #生成一字符串
```

```
cowboytype #获得索引位置 4 的对象
```

得到

```
'e'
```

在哪能找到：

第 12 章：lotto.py

sequence[initial:end]

切开表达式；切断序列的成员，起始于初始位置并且不到终点处停止。

例子：

```
comboytype[0:4]
```

得到

```
'Rhin'
```

在哪能找到:

第 10 章: sample.py

sequence[initial:]

切开表达式: 在初始处开始并到达序列的终点。

例子:

```
cowboytype[2:]
```

产生

```
'nestone'
```

在哪能找到:

第 10 章: lc.py

sequence[:end]

切开表达式: 在开始处启动并到达终端值。

例子:

```
cowboytype[:4]
```

返回

```
'Rhin'
```

在哪能找到:

第 12 章: lotty.py

sequence[:]

切开表达式: 产生完整序列。

例子:

```
cowboytype[:]
```

返回

```
'Rhinstone'
```

sequence*integer

序列乘以整数值的次数。实质上, 它是一个复制过程。

例子:

```
[5,6]*2
```

结果


```
[5, 6, 5, 6]
```

sequence+additional

连接序列而不管附加物是什么。这两个对象必须是同一类型的；例如，你不能连接一个列表和一个元组，也不能连接 `xrange()` 对象。

例子：

```
range(2)+range(5,7)#连接两个范围
```

返回

```
[0,1,5,6]
```

2.5.2 字符串

string%tuple

允许在字符串中使用占位符。从元组中引进可选数据并最终代替字符串中的每个占位符。有时，这在代码中对奇数寻找的字符串有利，但它能正常工作。除了 `%s` 以外，Python 有许多好的格式化代码。参考和软件包一起发行的 Python 文档。

例子：

```
results="Results: %s for the silly party, %s for the really silly lparty."
```

```
count=("50%", 0)
```

然后你可以用下面的式子进行合并：

```
results % count
```

这样会产生：

```
'Results: 50% for the silly lparty. 0 for the really silly party.'
```

在哪能找到：

第6章： `regexer.py`

第7章： `dbase3.py`

sequence%dictionary

允许在字符串中使用占位符。从字典中引进可选数据并最终代替字符串中的每个占位符。有时还有助于查找某些奇数字符串，但也能正常工作。这项操作更像邮件合并。除了 `%s` 以外，Python 有许多好的格式化代码。参考和软件包一起发行的 Python 文档。

例子：

```
data={"high": "72 degrees", "low": "14 degrees"}
```

```
report="The high today was %(high)s and the low was %(low)s."
```

```
report%data
```

将会返回

```
'The high today was 72 degrees and the low was 14 degrees.'
```

2.5.3 列表

列表在 Python 中有显著和可延展性。有许多操纵它们的方法。

del list[index]

del list[initial:end]

del list[index]在索引点删除元素。**del list[initial:end]**从初始索引值到刚好不到终点处删除元素。考虑一下另一种方法，这会删除一个片段。记住索引从 0 而不是 1 开始。

例子：

在索引点处删除列表项目

```
eric=["half", "a", "bee"] #生成一个列表
```

```
del eric[1] #删除第二个元素
```

```
eric #打印 eric
```

```
['half', 'bee']
```

删除列表项目的序列

```
eric=["half", "a", "bee"] #生成一个列表
```

```
del eric[0:2] #切断并删除元素 0 和 1
```

```
eric #打印 eric 的值
```

```
['bee']
```

在哪能找到：

第 6 章：dictEdit.py

list.append(element)

在规定的列表的尾端放置一个元素。这个列表本身被改变，而不是被复制。

例子：

```
eric=["half", "a", "bee"] #生成一个列表
```

```
eric.append("without license") #追加元素"without license"
```

```
eric #打印 eric 的值
```

```
['half', 'a', 'bee', 'without license']
```

list.count(element)

计算元素在列表中出现多少次。

例子：

```
albatross=["albatross!", "albatross!"] #生成一个列表
albatross.count("albatross") #查看该元素出现了多少次
正确地得到
2
```

list.extend(list)

把指针移到列表的尾端，并添加参数 list 的内容。

例子：

```
albatross=["albatross!", "albatross!"] #生成一个列表
parrot=["parrot", "parrot"]
albatross.extend(parrot) #扩充 albatross
albatross #列出 albatross 中的值
返回
```

```
['albatross!', 'albatross!', 'Parrot', 'parrot']
```

当然，这条语句改变了 albatross。为了保持原来的 albatross，你可以把语句改为：

```
albatrossparrot=albatross.extend(parrot)
创建一个全新变量 albatrossparrot。
```

list.index(element)

即使有多个 a，也会返回和 a 的第一个位置相符合的索引数。记住索引首先从 0 开始，而不是 1。

例子：

```
lotsanumbers=[1,2,3,4,2,6,4,7,8] #生成一个列表
lotsanumbers.index(2) #返回索引顺序中第一个 2 的索引
返回
1
```

list.insert(index,object)

在索引的位置处把对象插入列表。较高的元素就增加一个索引位置。

例子：

```
albatross=["albatross!", "albatross!"] #生成一个列表
albatross.insert(1, "bloody albatross!") #在索引 1 处插入对象
返回
['albatross!', 'bloody albatross!', 'albatross!']
```

list.sort()

list.sort(function)

`list.sort()`是这两个中比较简单的，并且它只根据适合于比较的标准 Python 规则进行排序。参考和 Python 包在一起的文档中的详细资料。`list.sort(function)`更加复杂但是更强大。`list.sort(function)`可以让你定义排序的函数，然后用 `function` 参数调用它，而不是使用 Python 的内置排序算法。

例子：

```
albatross=["albatross!", "bloody albatross!", " albatross!"] #生成一个列表
albatross.sort() #排序 albatross
albatross #打印 albatross 的值
返回以下的结果
[' albatross!', ' albatross!', 'bloody albatross!']
```

list.remove(element)

从列表中移除第一个 `element` 参数。如果在列表中有其他相同的元素，它们会适当保留。

例子：

```
albatross=[' albatross!',"bloody albatross!", " albatross!"] #生成一个列表
albatross.remove("albatross!") #删除第一次出现的元素
albatross #打印 albatross
得到
['bloody albatross!', ' albatross!']
```

list.reverse()

在列表中反转元素的排列次序。

例子：

```
revalbatross=['bloody albatross!', 'albatross!'] #生成一个列表
revalbatross.reverse() #反转该元素
revalbatross #打印 revalbatross 的值
返回
['albatross!', 'bloody albatross!']
```

2.5.4 字典

del dictionary[key]

删除关键字和它的数值。

例子:

```
del webster["number"] #删除其中的一个条目
webster #打印字典
{'size': 'normal', 'angle': 'sharp'}
```

dictionary[key]

使用关键字搜索条目。

例子:

```
webster={"number": "two", "size": "normal", "angle": "sharp"}
webster["size"]
'normal'
```

dictionary[key]=entry

把关键字和数值联合在一起。关键字必须是可散列化的。不是所有的数据形式都可以。列表就不可以；元组和字符串可以。

例子:

```
webster={"number": "two", "size": "normal", "angle": "sharp"}
webster["position"] = "prone" #生成新条目
webster #打印字典
返回
{'number': 'two', 'size': 'normal', 'angle': 'sharp', 'position': 'prone'}
```

dictionary.clear()

清除字典中所有的条目。

例子:

```
webster={"number": "two", "size": "normal", "angle": "sharp"}
webster.clear() #清除所有条目
webster #打印 webster 的内容
结果为
{}
```

dictionary.has_key[key]

检查字典中使用的关键字。如果使用了关键字返回真，如果没有使用则返回假。

例子:

```
webster={"number": "two", "size": "normal", "angle": "sharp"}
```

```
webster.has_key("size") #检查有效关键字  
得到
```

```
1
```

在哪能找到:

第 6 章: Andy 的窗口小部件

dictionary.items()

得到作为列表的关键字和数值, 列表中是元组而不是字典。

例子:

```
webster={"number":"two", "size":"normal", "angle":"sharp"}  
webster.items() #获取列表的关键字和值, 作为一列表  
返回  
[('number', 'two'), ('size', 'normal'), ('angle', 'sharp')]
```

dictionary.keys()

得到字典的所有关键字。

例子:

```
webster={"number":"two", "size":"normal", "angle":"sharp"}  
webster.keys() #获取所有关键字  
关键字返回如下  
['number', 'size', 'angle']
```

dictionary.values()

得到字典所有数值。

例子:

```
webster={"number":"two", "size":"normal", "angle":"sharp"}  
webster.values() #获取所有数值  
['two', 'normal', 'sharp']
```

len(dictionary)

返回字典中关键字/数值组合的数量。

例子:

```
webster={"number":"two", "size":"normal", "angle":"sharp"}  
len(webster) #获取字典组合数  
得到
```

2.6 文件

Python 的内部处理文件语句能打开文件，写文件并关闭文件。

`file.close()`

关闭命名文件。这条语句自动刷新文件，强制写入磁盘。释放文件的资源。

例子：

```
text.close() #关闭文件 text
```

在哪能找到：

第6章：regexer.py

`file.flush()`

使缓冲区的文件写入磁盘。当用 Python 写文件时，文件被超高速缓存而不是立即写入磁盘。`file.flush()`强制文件写入磁盘。

例子：

```
text.flush() #将 text 写入磁盘
```

`file.isatty()`

试图查看是否文件将来自磁盘，或者来自键盘/终端设备。`tty` 返回真。

例子：

```
text.isatty() #检查 tty 来源
```

返回

```
0
```

检查设备

```
inf = open("/dev/tty", "r")
```

```
inf.isatty()
```

```
1
```

`file.read()`

`file.read(number of characters)`

读取前面已经打开的文件（意思是现在它是一个对象）。参阅 `open()`。`file.read()`从当前位置开始到文件的结尾读取整个文件。`file.read(number of characters)`只读取从当前位置指定的字符的数目。如果到达文件的结尾没有遇到可读取的字符，Python 返回 `None` 并且输出是

“ ”。参阅 `file.seek()`。

例子:

```
text=open("test.txt", "r+") #用读/写模式打开文件
text.read() #读取文件内容
```

在哪能找到:

第 7 章:dbase3.py

`file.readline()`

从文件中读取一行。如果到达文件的结尾没有找到可读取的字符。Python 返回 `None`, 显示为 “ ”。参阅 `file.seek()`。

例子:

```
text=open("text.txt", "r+") #打开文件
text.readline() #从 text 中读取第一行
```

`file.seek(number)`

`file.seek(number,position)`

`file.seek(number)` 移动指针到文件中相对于第一个字节（索引零）的位置。`file.seek(number,position)`将指针移动到文件中相对于指定位置的位置。0 意味着相对于零的位置, 1 意味着相对于当前位置, 2 意味着相对于文件的结尾。指定除文件的最后一个字节之外的字节的位置是可能的, 但 Python 不能超过 EOF（文件的结尾）标志。参阅 `file.tell()`。

例子:

```
text.seek(55) #指向文件末尾
text.seek(-10, 2) #从文件尾前移 10 个字符
```

在哪能找到:

第 7 章: dbase3.py

`file.tell()`

告诉你目前在文件中的位置。计数通常从零开始, 而不是 1。

例子:

```
text.tell() #找出文件的当前位置
告诉你目前的位置
```

45

`file.truncate`

在当前位置截断文件。

例子:

```
outf = open("sample.txt", "r+")
outf.write("Truncate this!\n")
outf.seek(9)
outf.tell()
9
outf.truncate()
outf.seek(0)
outf.read()
'Truncate'
```

file.write(string)

在当前定位 (seek) 的位置之后写一个字符串。如果对象指针在文件的开始处, 原来的字节就被冲掉改写。要完全写入文件, 文件必须已经被打开并具有写权限。参阅 open()、file.seek()。

例子:

```
text=open("test.txt", "r+") #打开文件
text.tell() #获取最后一个字符的位置
返回
55
然后可以写入
text.seek(55) #将指针移到文件末尾
text.write('this is appended') #将该元素置于文件末尾
在哪能找到:
第7章: database3.py
```

2.7 异常 (Exception)

当告诉 Python 去做它不愿意做的事情时就会引发异常: 试图执行有关错误类型的对象操作、溢出内存、寻找在不存在的东西和许多其他事情时的操作。你可以定义自己的异常, 甚至有故意引发异常的能力。

Python 有内置异常的长列表, 而且模块也有自己的异常。异常是对象、字符串或类对象。

2.7.1 异常的列表

AssertionError

无论你是否通过请求最优化启用调试模式，它都允许插入有条件的调试语句。参阅 `assert`。

在表达式可测试的地方，这条语句采用 `assert` 表达式的形式。只有当内置 `__debug__` 变量为 1 时才有用。当你请求最优化时该变量成为 0。

例子：

```
assert 1>2 #生成一个错误
```

在交互模式中，并且设置了 `_debug_` 后，给出

```
Traceback (innermost last):
File "<stdin>", line1, in?
AssertionError
```

AttributeError

当 Python 不能找到属性或属性没有正确赋值时就会引发 `AttributeError`。如果属性可能不适合于对象类型，就会引发 `TypeError`。参阅 `NameError`、`TypeError`。

例子：

```
pumpedup.downer #试图使用一个不存在的属性
只能得到
Traceback (innermost last):
File "<stdin>", line1, in?
AttributeError:downer
```

FloatingPointError

当浮点操作失败时，就会引发 `FloatingPointError`。只有当 Python 用 `-with-fpectl` 选项进行配置，或者 `WANT_SIGFPE_HANDLER` 符号在“`config.h`”文件中定义时，这个异常总是被定义。

IOError

当你试图用 I/O 做一些不能做的事情时就会出现 `IOError`。当你试图打开不存在的文件或写入不能存取文件的媒介时，通常就能看见这个异常。

例子：

```
open("snothere.txt") #试图打开一不存在的文件
```

当然结果为

```
IOError: [Errno 2] No such file or dictionary: 'snothere.txt'
```

ImportError

当试图输入一些不能找到的文件时就会引发 `ImportError`。应用于模块或模块的方法。

例子:

```
import snothere #试图导入不存在的内容
```

Python 相应地响应

```
ImportError: No module named snothere
```

IndexError

当请求超出范围的索引位置时,就会让你知道发生了 `IndexError`。注意当超出范围时,切片表达式会自动解决并且不引发异常。

例子:

```
sequence=[1,2,3] #生成一个只有三个元素的序列
```

```
sequence #请求查看第四个元素
```

返回

```
IndexError: list index out of range
```

KeyError

当请求实际上不在字典中的字典关键字时就会引发 `KeyError`。

例子:

```
little_dictionary={"this":"that", "far":"near"} #生成一个字典
```

```
little_dictionary["up"] #查找该字典中不存在的关键字
```

得到

```
KeyError: up
```

KeyboardInterrupt

当 Python 检测 `interrupt` 键的操作时就会引发。

例子:

```
input("Your name's not Bruce?") #从键盘输入
```

如果用户按 `interrupt` 键 (根据系统而改变), 就会发生:

```
KeyboardInterrupt
```

MemoryError

当内存溢出时在个别情况下会引发。这不是一个经常发生的问题，因为多数脚本是一些小程序，而且现代计算机则有几乎无限的内存。但这有可能发生，多半是在无限循环的情况中和对象创建时发生。

NameError

Python 不能找到你请求的对象时会通知你。当你忘记输入模块但又尝试调用它的函数时，这个错误就会发生。

例子：

```
aint-there("wha?") #调用不在名字空间中的函数
```

返回

```
NameError: aintthere
```

NotImplementedError

有点深奥，这个异常只有在用户自定义的基类时出现。查看和你的软件包在一起的 Python 文档中的详细资料。

OSError

虽然 `OSError` 是内置异常，它主要与从 `os` 模块返回的操作系统错误一起运转。这个异常适用于在 Python 工作空间的友好的区域的外部发生的事情。

OverflowError

表示你试图执行太大的算术运算。不从长整数引发，而且很少从浮点数引发。

例子：

```
979797979797 * 979797979797 #两个大数相乘
```

得到

```
OverflowError: integer literal too large
```

RuntimeError

在一个错误没有被另一个明确的异常覆盖时发生。不常使用，多数是前一 Python 版本带过来的。

SyntaxError

违反语法时引发。你可以在几种情况下得到语法错误：当在交互模式中时，当用 `exec()`

或 `eval()` 运行时，当使用 `input()` 时或者当脚本第一次被读取时。

例子：

```
def grail(cup=None) #忘记了冒号
```

Python 会提示你出现问题的地方：

```
File "<stdin>", line 1
```

```
def grail(cup=None)
```

```
SyntaxError: invalid syntax
```

屏幕上，在 `(cup=None)` 中的最后一个括弧的下面，有一个标记可以让你知道最后一个有效字符的位置。

SystemError

罕见的异常。由内部错误产生。尝试生成字符串的注释，这样你就能报告这种异常。

SystemExit

由 `sys.exit()` 函数产生的一种异常。当它发生时，Python 接近停止并退出。有许多 Python 程序员用下面的表达式来终止一个会话：

```
raise SystemExit
```

TypeError

当试图执行错误类型的操作时就会引发。

例子：

```
carpool=['Fred', 'Sally', 'Dr. Graham'] #生成一个列表对象
```

```
carpool['Fred'] #在其上执行一字典操作
```

这点错误会导致：

```
TypeError: sequence index must be integer
```

ValueError

当运算指令得到一个有效但不正确的参数时就会弹出该错误，并且当这个错误不包含在异常中时，它就更具有可读性。

例子：

```
carpool=['Fred', 'Sally', 'Dr. Graham'] #生成一列表对象
```

```
carpool.remove('Dr. Graham')
```

```
ValueError: list.remove(x): x not in list
```

ZeroDivisionError

当你试图用零除时就会出现这种错误。这个异常经常发生在没有预料到被零除的可能、不经意地被零除，或者在试图分开其他文字时一不小心键入 0。

例子：

```
crashme=oops/0
```

当 Python 试图执行这条语句时，除了向后跟踪以外，你几乎无疑会得到：

```
ZeroDivisionError: integer division or modulo
```

其异常也就有这么多了。

2.7.2 建立自己的异常

Python 有两种定义异常的类型：字符串和对象。字符串异常不过是当条件满足时，你生成的字符串会弹出来。如果你正在检测不合适的拼写检查模式的使用，例如，你可以有一个脚本用字符串给你发出警告。另一种可选择的方法是，你能用类语句定义更灵活、更强大的异常。

2.7.3 处理异常

程序员最困难的任务之一是在程序被很多的粗心大意的人执行之后，预见在哪个地方出现了问题。编制异常处理的代码也是很棘手的。尽管 Python 有处理可预言的异常的极好的方法。不可预见的异常仍然损害了程序的中间步骤，但你能漂亮地处理你所预料的任何事。

Raise

允许你定义并引发自己的异常，或者触发任意你喜欢使用的内部异常。

例子：

定义一个异常：

```
CowError="Moo!" #设置字符串异常 CowError
```

```
def cowfly(condition):
```

```
    if condition == "walking":
```

```
        raise CowError, "Wrong skit!"
```

```
    return condition
```

当像下面这样调用 cowfly() 时：

```
cowfly('walking')
```

你自己的异常字符串生成的外观如下：

```
Moo! Wrong skit!
```

要调用内置错误：

```
raise TypeError, "you're not my type, you know"
```

在交互模式下, 得到

```
TypeError: You're not my type, you know
```

在哪能找到:

第10章: `ebcdic.py`

第11章: `python2c.py`

try/except

try/finally

捕获异常。如果有理由相信某些异常可以引发, 当它们出现时, 你能规定代码做什么。

用 `try` 语句包括有问题的代码。有两种风格的 `try` 语句: `try/except` 和 `try/finally`。在 `try` 语句之后的 `try/except` 语句有许多行, 这些行具有生成一个或多个异常的能力。在 `try` 语句下的每条 `except` 语句都能容纳一个特殊的异常, 以及当异常触发时你想做的事情。没有明确异常的 `except` 语句会触发所有的异常。你可以在“异常链”结尾处随意放置一条 `else` 语句, 来指定如果没有一个异常触发时会发生什么。

`try/finally` 语句主要用于清除。例如, `finally` 语句放在释放系统资源的代码之前。

虽然可以使用这两种格式, 但不能将这两种混合在一起用。

例子:

```
def dangit(): #定义一个函数
```

```
    try:
```

```
5/0 #对这个表达式使用 try 语句
```

```
    except ZeroDivisionError: #它是否是这个异常?
```

```
print "Dang it! Division by zero again!" #打印此内容
```

```
dangit() #调用它
```

现在, 解释程序打印出:

```
Dang it! Division by zero again!
```

在哪能找到它:

第10章: `lc.py`

第11章: `tabfix.py`

在一条子句中, 你能捕捉若干 `exception`。例如, 你可以这样做:

```
except (ValueError, IndexError):
```

第 3 章 模块

本章要点:

- Python 服务
- 字符串服务
- 其他服务
- 普通操作系统服务
- 可选操作系统服务
- UNIX 特定服务
- CGI 和 Internet
- 限制性运行
- 多媒体
- 加密服务
- SGI IRIX 特定服务
- SunOS 特殊服务
- Microsoft Windows 特殊服务

Python 的真正的魔力存在于提供模块的集合。模块给了 Python 可扩展的范围，有一切种类操作的模块。另外，你能修改许多提供的模块，甚至自己编写模块。本章列出了所提供的模块和它们的函数。参见 Python 文档得到更多的详细资料。

模块函数只有在把它们输入 Python 名字空间之后才能被调用。参见第 2 章的“输入”，得到更多关于怎样输入模块的详细资料，或者参见 Python 文档。

某些模块做许多和第 1 章所讲的 Python 内部操作符一样的工作，只是模块可以有一些变化或改进。模块按它们在 Python 文档出现的顺序显示。参考 Python 文档可以得到它们的完整信息。它们在这里只是作为脚本的快速参考显示。大部分说明都直接由 Python 文档产生。我将在各处增加一点儿。

记住不同的 Python 发行版本有不同的模块，即使文档是一样的。例如，如果你使用 Windows Python 发行版本，你得不到 UNIX 专用的模块（如果你在用 Windows 平台，请到 www.linux.com 核对以更好地使用它）。

3.1 Python 服务

sys

特定的系统参数和函数。

在哪能找到：

第7章：dbases3.py

第9章：engine.py

第9章：who-owns.py

第10章：banner.py

第10章：lc.py

第11章：spinner.py

第11章：tabfix.py

第11章：space.py

types

所有内置类型的名字。

在哪能找到：

第6章：Andy 的窗口小部件

UserDict

字典对象的类包装器。

UserList

列表对象的类包装器。

operator

作为函数的标准操作等。

traceback

打印或获取栈的跟踪结果。

在哪能找到：

第9章：sengine.py

pickle

Python 对象序列化。Pickling 指的是把对象转换为能通过网络存储或发送的位流。你能序列化整数、长整数、浮点数、字符串、元组、列表和只包含可序列化对象的字典。在编组上序列化的一个好处是你能序列化自定义对象。

cPickle

`pickle` 的替换实现版本，只是它用 C 语言实现，用起来快得多。

copy_reg

注册 `pickle` 支持函数。

shelve

Python 对象持续性。

在哪能找到：

第 7 章：nickname.py

copy

浅的和深的复制操作。不同之处是浅的 `copy` 生成现有复合对象之外的一个新的复合对象，并引用原始对象放入这个副本中。深的 `copy` 实际上拷贝每个对象。

marshal

Python 对象序列化（具有不同的限制条件）。`marshal` 读取 Python 数值，并把它写成独立系统二进制格式。你能序列化整数、长整数、浮点、字符串、元组、列表、字典和代码对象。元组和字典只有当支持它们的内容时它们才被支持。

imp

访问 `import` 语句的实现。

parser

访问 Python 代码的分析树。

symbol

表示分析树的内部节点的常量。

token

表示分析树的终端节点的常量。

keyword

测试字符串是否是 Python 的关键字。

code

代码对象服务。

pprint

数据优质打印机。本质上，它用格式化的某些外观（semblance）打印数据。

repr

另一种 repr() 的实现。

py_compile

编译 Python 源文件。

compileall

对 Python 库文件进行字节编译。

dis

反汇编程序。当然，没有任何真正的 Python 汇编程序，但是这种模块翻译了解释程序运行的字节码。

site

引用特定位置模块的标准方法。和大多数模块不一样，它被自动输入。

user

引用特定用户模块的标准方法。user 允许你设置定制文件，当它初始化时脚本自动访问。

__builtin__

内置函数。

`__main__`

最高层次的脚本环境。

3.2 字符串服务

`string`

普通字符串操作。如果你做过许多字符串操作，你应该对这个模块很熟悉。

在哪能找到：

第 10 章：`strplay.py`

第 10 章：`xref.py`

`re`

Perl 风格规则表达式操作。

在哪能找到：

第 8 章：`roman.py`

第 10 章：`xref.py`

第 11 章：`tree.py`

`regex`

规则表达式的搜索并匹配操作。自从 Python 1.5 以来就考虑废弃这个模块，并且不应该再用于新的脚本。虽然，旧的脚本也许还仍然使用它。如果可能的话，Python 文档建议你
把 `regex` 移植成 `re`。参阅 `re`。

在哪能找到：

第 6 章：`regexer.py`

`regsub`

使用规则表达式的替换运算和分割运算。自从 Python 1.5 以来就考虑废弃这个模块。如果可能的话，Python 文档建议你使用 `re` 来代替它，并且把现有脚本的 `regsub` 移植成 `re`。参阅 `re`。

`struct`

像压缩的二进制数据一样解释字符串。这个模块执行 Python 到由 Python 字符串表示的 C 的结构体的转换。

StringIO

读取并编写字符串，就好像它们是文件一样。

cStringIO

StringIO 的更快的版本，但不可以有子类。

3.3 其他服务

math

数学函数。这是另一个非常非常有用的模块。如果你做大量的数学编程（谁不这么做呢），你会愿意和 math 模块交个好朋友。

在哪能找到：

第 8 章：donutil.py

第 8 章：stats.py

cmath

复杂数字的数学函数。

whrandom

浮点伪随机数字发生器。

在哪能找到：

第 10 章：sample.py

第 11 章：otp.py

第 12 章：logic.py

random

生成具有不同分布的伪随机数。

bisect

按排序次序维护列表的有效方式。

array

支持数组。Python 自然不支持传统数组。这个模块中的数组像列表一样进行操作，除

了列表中的对象是被键入的以外。通常，Python 不能让你声明一个类型；这个模块就可以。

ConfigParser

配置文件语法分析程序。让终端用户定制你的脚本。

fileinput

从多重输入流中反复操作行。

calendar

仿真 UNIX cal 程序的函数。

cmd

构建面向行的命令解释程序。

shlex

简单词法分析。

3.4 普通操作系统服务

OS

其他 os 接口让你使用一个模块（就是这个模块），而不是挑选一个特定的 os 模块。某些函数是系统特定的，所以检查 Python 文档，看看哪一个是通用的，哪一个是特定的模块。

在哪能找到：

第 11 章：space.py

第 11 章：tree.py

os 模块给了你使用或多或少的熟悉的命令，在目录结构中来回走动的能力。例如，你可以使用 os.chdir() 改变目录，os.getcwd() 获取你当前指向的目录的名字。然而，要记住使用这段代码能限制脚本的可移植性。

os.path

普通路径名操作。

time

时间访问和转换。Python 没有内置 time 函数，所以这个模块在使用 time 函数的脚本中

是必不可少的。

在哪能找到：

第9章：sengine.py

getpass

可移植口令的读取。提供读取不回现口令的函数，并获取用户的登录名。

getopt

命令行选项的语法分析程序。类似于 UNIX 的 getopt() 函数。

tempfile

生成临时文件名。它实际上没有创建文件；你必须创建这个文件。

errno

标准 errno 系统符号。

glob

UNIX shell 的路径名搜索。

fnmatch

UNIX shell 的风格路径名的模式匹配。

shutil

高级文件操作。

locale

国际化服务。让你把文化方面（例如，小数点位置和流通货币）的情况并入到你的脚本中。

3.5 可选的操作系统服务

signal

为异步事件设置处理器。

socket

低层次的网络接口。

select

在多重流上等待 I/O 完成。

thread

多重控制线程。

在哪能找到：

第 9 章：sengine.py

第 9 章：engine.py

threading

较高级别的线程接口。

Queue

同步队列类。

anydbm

DBM 风格数据库模块的普通接口。

dumbdbm

简单 DBM 接口的可移植实现。

whichdb

推测给定数据库创建的 DBM 风格模块。

zlib

与 gzip 兼容的压缩和解压缩。

gzip

使用文件 gzip 压缩和解压。

3.6 UNIX 特定服务

posix

最通用的 POSIX 系统调用。Python 文档不同意直接输入这个模块，所以不要这样做。相反，输入 `os` 并让 `os` 做这项工作。

pwd

口令数据库。用于访问 UNIX 口令数据库。

grp

提供对 UNIX 组数据库的访问。

crypt

执行用于检测 UNIX 口令的 `crypt()` 函数。

dbm

基于 `ndbm` 的标准“数据库”接口。

gdbm

`dbm` 的 GNU 不同解释。

termios

POSIX 风格的 `tty` 控制。

TERMIOS

和 `termios` 模块一起使用的常量。

fcntl

`fcntl()` 和 `ioctl()` 系统调用。

posixfile

支持锁定的像文件一样的对象。这个文档警告“在将来的版本中”将会废弃这个模块，并且力劝你用 `fcntl` 代替它。参阅 `fcntl`。

resource

资源使用信息。它可以让你限制资源。这个特定资源根据你所使用的系统的不同而不同。

syslog

UNIX syslog 库例程序的接口。

stat

解释 stat()结果的实用工具。

popen2

可访问标准 I/O 流的子进程。让你产生进程并把它们的管道连接在一起。

commands

os.popen()的包装函数。

3.7 CGI 和 Internet

cgi

支持 CGI 脚本的模块。用输入语句“import cgi”而不是“from cgi import”来使用它。

在哪能找到：

第 9 章：sengine.py

urllib

打开 url 给出的随机对象。本质上，它就是 Web 的 open()函数。

在哪能找到：

第 9 章：sengine.py

httplib

HTTP 协议客户程序。通常不直接使用，而是作为对 urllib 的支持使用。参阅 urllib。

在哪能找到：

第 9 章：sengine.py

ftplib

FTP 协议客户程序。

gopherlib

Gopher 协议客户程序。

poplib

POP3 协议客户程序。

imaplib

IMAP4 协议客户程序。

smtplib

SMTP 协议客户程序。

urlparse

把 URL 分列成组件。

SocketServer

网络服务器的结构。

BaseHTTPServer

基本 HTTP 服务器。典型地，这个模块不直接使用，而是作为服务器的基础使用。

htmllib

HTML 文档的分析程序。它的 `HTMLParser` 类作为其他类的基类使用。`HTMLParser` 本身是 `sgmlib` 中的 `SGMLparser` 的一个子集。这个模块对于 HTML2.0 有效。

xmllib

XML 文档的分析程序。当 XML 技术被不同的双亲使用时也许需要一些调整。定义一个类 `XMLParser`。这个模块支持 XML 名字空间，即使在编写 XML 名字空间时，它也只是个提议，而不是一个标准。

formatter

普通输出格式程序和设备接口。由 `htmlib` 模块的 `HTMLParser` 类使用。

rfc822

分析 RFC822 邮件头。这个模块中的类 `Message` 能处理大部分 RFC822 邮件头。

mimertools

分析 MIME 风格消息体的工具。

MimeWriter

普通 MIME 文件编写程序。`MimeWriter` 类是创建 MIME 文件的基本格式化程序。

multifile

支持读取包含截然不同部分的文件。

binhex

用 `binhex4` 格式编码和解码文件。为了与 Mac 的兼容性。

uu

用 `uuencode` 格式编码和解码文件。

binascii

在二进制和不同编码的 ASCII 表示之间进行转换。

xdrlib

编码和解码 XDR 数据。支持外部数据表示（External Data Representation）标准。

mailcap

`mailcap` 文件处理。

mimetypes

映射文件扩展名到 MIME 类型。

base64

编码和解码 MIME base64 编码技术。

quopri

编码和解码 MIME 引用适于打印的编码技术。

mailbox

读取不同的邮箱格式。用于 UNIX 邮箱。

mimify

mimi 化和反 min 化邮件消息，是邮件信息到 MIME 以及相反的简单转换。

netrc

netrc 文件处理。为 ftp 客户程序处理 netrc 格式。

3.8 限制运行

在正常情况下，Python 对操作系统进行完全访问。当然，这样既有益处也很危险。能够直接抓取文件当然好，但你也许不想让外来者登录你的机器从而对整个系统进行访问。结合运行可以让你编写不对系统进行完全操作的脚本。而是改为，依次创建和系统相互作用的环境。它是有点简陋的防火墙。当置信和非置信时，Python 文档引用两种不同的脚本。

rexec

基本限制运行框架。有 rexec 类，它限制了 exec()、execfile()、eval()和 import()的模式型式。

Bastion

提供对对象的限制访问，总是和 rexec 模块一起使用。允许或禁用对对象属性的访问。

3.9 多媒体

audioop

操作未加工处理过的音频数据。处理作为 Python 字符串存储的有符号的整数取样。

imageop

操作未加工处理过的图像数据。图像类型的限制范围：在 Python 字符串中存储的 8 位

或 32 位像素图形。

aifc

用 AIFF 或 AIFC 格式读取和编写的音频文件。

rgbimg

用 SGI RGB 格式读取和编写图像文件。有限的功能性但却很有用。

imghdr

确定图像格式。能区别 RGB、GIF、PBM、PGM、PPM、TIFF、RAST、XBM、JPEG、BMP 和 PNG 格式。

sndhdr

确定声音文件格式。

3.10 加密服务

你可以有也可以没有这些模块，根据所使用的 Python 包来决定。

md5

用 MD5 消息摘要算法创建一个 MD5 对象。

在哪能找到：

第 11 章：otp.py

mpz

任意精度运算的 GNU MP 库。你需要 GNU MP 软件使这个模块进行工作。

rotor

像 Enigma 一样的加密和解密。只对有点模仿在第二次世界大战中德国使用的 Enigma 编码/解码机器的很少的模块感兴趣。有足够的安全性，从而德国人也认为它们最初的 Enigma 机器是难以侵入的，直到战争结束，英国已经侵入它了，他们还是一直没有人知道。

3.11 SGI IRIX 特定服务

这些模块在 IRIX 的第 4 和第 5 版本中使用。

al

SGI 上的音频函数。

cd

SGI 上的 CD-ROM 接口。

fl

GUI 应用程序的 FORMS 库接口。

flp

存储 FORMS 设计的加载函数。

fm

SGI 工作站的字体管理程序接口。这是一个便利的模块，不是一个完整的模块。

gl

来自 SGI 图形库的函数

DEVICE

和 gl 模块一起使用的常量。

imgfile

支持 SGI imglib 文件。

jpeg

用 JPEG 格式读取并编写图像文件。

3.12 SunOS 特定服务

适用于 SunOS 的第 4 和第 5 版本。

sunaudiodev

访问 Sun 音频硬件。

3.13 Microsoft Windows 特定服务

msvcrt

来自 MS VC++ 运行库的有用的例行程序。

winsound

适用于 Windows 的声音播放接口。

第 4 章 Tkinter

本章要点：

- 窗口小部件

Tkinter 是 Python 默认的 GUI（图形用户接口）。Python 没有自己的 GUI，所以用 Tkinter 来代替。但是，如果你想要做实验的话，其他 GUI 也是可用的。

Tcl/Tk 最初是由 U.C.Berkeley 的 John Ousterhout 博士开发的。他后来迁职到 Sun Microsystems 公司，而如今在 Scriptics。Tkinter 是一个和 Tk 接口的 Python 模块，一个由 Ousterhout 开发的图形化工具包。目前，Tcl/Tk 可以从 www.scriptics.com 上的 Scriptics 中得到。为了使用 Tkinter，就像输入任何模块一样输入它。参见第 2 章的输入以得到更多的详细资料。

虽然 Python 和 Tkinter 兼容，但它不完全和 GUI 世界兼容。用 Python 或某些其他语言编写的脚本通常被意指为只用一次的程序。你调用它们，它们就做一项工作，然后离开。相反，GUI 应用程序是事件驱动，意味着它们建立一个等待用户单击、键入或移动某个东西的循环。当这种事情发生时，应用程序与事件互作用，然后掉进“用后门刺激狗”的状态，急切地等待你做其他事情。

4.1 窗口小部件

Tkinter 中的几乎每件东西都是一个窗口小部件。按钮、字段和窗口，全部都是窗口小部件。并且窗口小部件也都是具有属性的对象。查看 Tkinter 文档以得到详细资料。

4.1.1 Button（按钮）窗口小部件

在窗口中显示普通的按钮。参阅 Checkbutton 窗口小部件和 Radiobutton 窗口小部件，如图 4-1 所示。

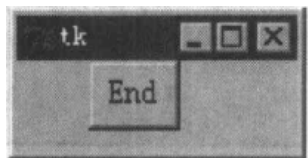


图 4-1 按钮窗口小部件

例子：

```
endbutton = Button(parent, text="End", command=self.ok)
endbutton.pack()
```

4.1.2 Canvas (画布) 窗口小部件

引起所有的其他 canvas 窗口小部件。一个令人讨厌的窗口小部件，因为它不是直接可见的，而它又是必不可少的。像其他的窗口小部件一样，它必须被显示地创建。

例子：

```
root=Tk()
drawingsurface=Canvas(root, width=100, height=100)
drawingsurface.pack()
```

4.1.3 Canvas Arc (画布弧) 项目

绘制一个扇形图、弦或者真正的弧，如图 4-2 所示。

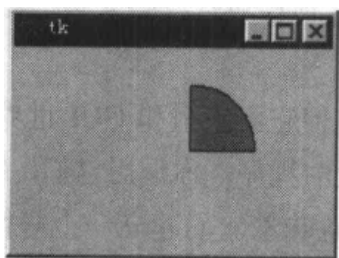


图 4-2 canvas arc 项目

例子：

```
root=Tk()
drawingsurface=Canvas(root, width=100, height=100)
drawingsurface.create_arc(5, 5, 90, 90, fill="red")
drawingsurface.pack()
```

4.1.4 Canvas Bitmap (画布位图) 项目

在画布上绘制一个位图。你可以使用自己的 XBM 图形或内置图形之一。后者包括沙漏光标、“i”表示信息、问号和警告图标，如图 4-3 所示。

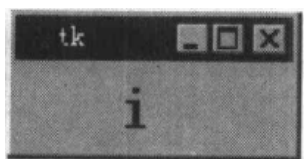


图 4-3 canvas bitmap 项目

例子：

```
root=Tk()
```

```
drawingsurface=Canvas(root, width=100, height=100)
drawingsurface.create_bitmap(50, 50, bitmap="info")
drawingsurface.pack()
root.mainloop()
```

4.1.5 Canvas Image（画布图像）项目

在画布上创建一个图像项目，在画布上你可以放置 GIF、PPM 或 PGM 图像，如图 4-4 所示。注意这是两个步骤。首先你必须创建一个 PhotoImage 类的成员；然后必须明确地把它引进画布中。

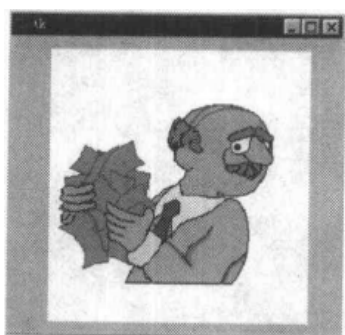


图 4-4 canvas image 项目

例子：

```
root=Tk()
drawingsurface=Canvas(root, width=200, height=200)
photo=PhotoImage(file="brian.gif")
canvasphoto=drawingsurface.create_image(10,10, anchor=NW, image=photo)
drawingsurface.pack()
canvas.pack()
root.mainloop()
```

4.1.6 Canvas Line（画布线条）项目

在画布上绘制线条，如图 4-5 所示。

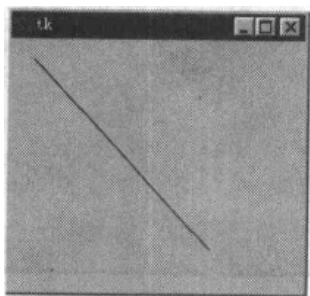


图 4-5 canvas line 项目

例子:

```
root=Tk()
canvas=Canvas(root, width=200, height=200)
whatsmyline=canvas.create_line(10, 10, 100,100)
canvas.pack()
root.mainloop()
```

4.1.7 Canvas Oval (画布椭圆) 项目

在画布上放置椭圆。椭圆符合决定它的形状的边界框的内侧，如图 4-6 所示。

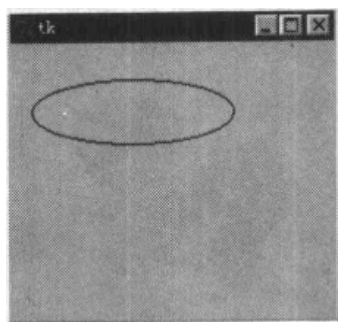


图 4-6 canvas oval 项目

例子:

```
root=Tk()
canvas=Canvas(root, width=200, height=200)
oval=canvas.create_oval(10, 10, 100, 50)
canvas.pack()
root.mainloop()
```

4.1.8 Canvas Polygon (画布多边形) 项目

在 canvas 上绘制多边形，如图 4-7 所示。因为大自然通常并不是圆形或正方形，所以多边形对于绘制逼真的形状派得上用场。

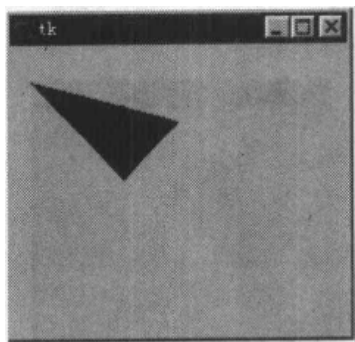


图 4-7 canvas polygon 项目

例子:

```
root=Tk()
canvas=Canvas(root, width=200, height=200)
polygon=canvas.create_polygon(10, 10, 100, 50, 70, 100)
canvas.pack()
root.mainloop()
```

4.1.9 Canvas Rectangle (画布矩形) 项目

在画布上放置矩形, 如图 4-8 所示。如果你想要查看适用于其他诸如椭圆形状的边界框, 矩形是很有帮助的, 例如椭圆。

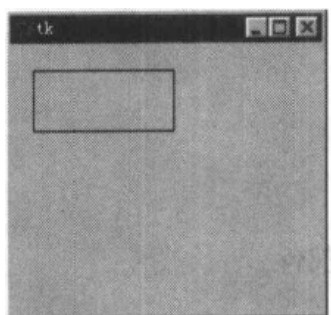


图 4-8 canvas rectangle 项目

例子:

```
root=Tk()
canvas=Canvas(root, width=200, height=200)
rectangle=canvas.create_rectangle(10, 10, 100, 50)
canvas.pack()
root.mainloop()
```

4.1.10 Canvas Text (画布文本) 项目

在 canvas 上绘制文本, 如图 4-9 所示。

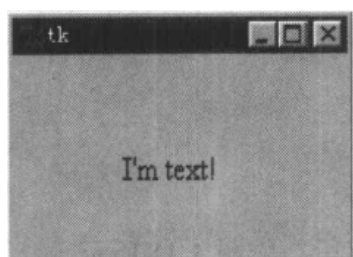


图 4-9 canvas text 项目

例子:

```
root=Tk()
```

```
canvas=Canvas(root, width=200, height=200)
text=canvas.create_text(100, 100, text="I'm text!")
canvas.pack()
root.mainloop()
```

4.1.11 Canvas Window（画布窗口）项目

在画布中插入一个窗口。默认地，窗口项目扩展到窗口的尺寸。

例子：

```
root=Tk()
canvas=Canvas(root, width=200, height=200)
window=
canvas.create_window(100, 100)
canvas.pack()
root.mainloop()
```

4.1.12 Checkbutton 窗口小部件

在窗口中放置一个复选框，如图 4-10 所示。当然，这是一个有两种状态的经典复选框：复选和非复选。

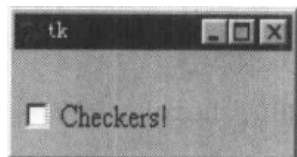


图 4-10 checkbutton 窗口小部件

例子：

```
root=Tk()
var=IntVar()
checkboxbutton1=Checkbutton(root, text="Checkers!", variable=var)
checkboxbutton1.pack()
root.mainloop()
```

4.1.13 Entry 窗口小部件

在窗口中插入一个文本输入字段，如图 4-11 所示。

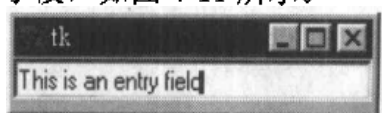


图 4-11 entry 窗口小部件

例子:

```
root=Tk()  
entry=Entry(root)  
entry.pack()  
root.mainloop()
```

4.1.14 Frame（框架）窗口小部件

在窗口中创建一个框架，在框架中可以放置图形、视频和其他东西。它也起着在窗口小部件之间连接的作用。

例子:

```
root=Tk()  
frame=Frame(root, width=100, hight=100, bg="red")  
frame.pack()  
root.mainloop()
```

4.1.15 Label（标签）窗口小部件

在窗口中放置预定义文本区，如图 4-12 所示。标签窗口小部件是对 Python 版本的“Hello World”编码最快的方法。

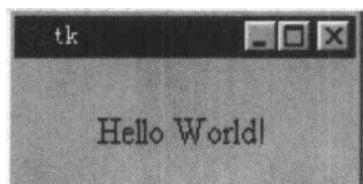


图 4-12 label 窗口小部件

例子:

```
root=Tk()  
label=Label(root, text="Hello world!")  
label.pack()  
root.mainloop()
```

4.1.16 Listbox（列表框）窗口小部件

插入可能选择的列表框，如图 4-13 所示。这个窗口小部件不是你在其他 GUI 惯例中熟悉的下拉（组合）框。这个例子只显示了怎样创建一个列表框，而不是怎样从列表框中获取结果。

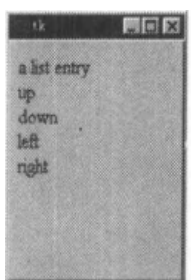


图 4-13 listbox 窗口小部件

例子:

```
root=Tk()
listbox=Listbox(root, relief=RAISED)
listbox.insert(END, "a list entry")
for item in ["up", "down", "left", "right"]:
    listbox.insert(END, item)
listbox.pack()
root.mainloop()
```

4.1.17 Menu (菜单) 窗口小部件

在窗口或框架上放置菜单，如图 4-14 所示。通常和 menubutton (菜单按钮) 窗口小部件一起使用。对于菜单单独使用的例子，参见附赠 CD-ROM 上的 Grail 的代码。



图 4-14 menu 窗口小部件

菜单被制成“可按虚线撕下的纸片”，这样使单击菜单并进入它自己的小窗口成为可能，如图 4-15 所示。



图 4-15 menu 窗口小部件示例的结果

例子:

```
from Tkinter import *
frame = Frame()
label = Label(width=48, text="Wot'll it be, then?")
menubar = Frame(frame, relief="raised", bd=2)
```



```
sampleMenu = Menubutton(menubar, text="File")
pane = Menu(sampleMenu)
for menuitem in ["Spam", "Spam", "Spam", "Eggs", "Spam"]:
    # 为这个菜单项定义回调函数
    def menuItemCallback(statusLabel=label, newMsg=menuitem):
        statusLabel['text'] = "Can I have her %s? I love it!" % newMsg
    pane.add_command(label=menuitem, command=menuItemCallback)
    pane.add_separator() # 添加一水平分隔线
pane.add_command(label="Exit", command=frame.quit)
sampleMenu['menu'] = pane
sampleMenu.pack(side="left") # 从左向右排列下拉菜单
right.
menubar.pack(side="top", fill="x", expand="n")
label.pack(side="bottom", fill="both", expand="y")
frame.pack(fill="both", expand="y")
frame.wininfo_toplevel().title("The Spam Sketch")
frame.mainloop()
```

4.1.18 Message（消息）窗口小部件

显示文本行，如图 4-16 所示。`entry` 窗口小部件只适合单行的情况；但 `message` 窗口小部件却可以用于多行。另外，它还有其他有趣的事情需要你阅读 Tkinter 文档去发现。

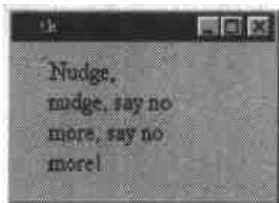


图 4-16 message 窗口小部件

例子：

```
root=Tk()
message=Message(root, text="Nudge, nudge, say no more, say no more!")
message.pack()
root.mainloop()
```

4.1.19 Radiobutton（单选按钮）窗口小部件

如图 4-17 所示，具有按钮和复选按钮，构成了 Tkinter 中的基本输入机制。参阅

Checkbutton 窗口小部件和 Button 窗口小部件。

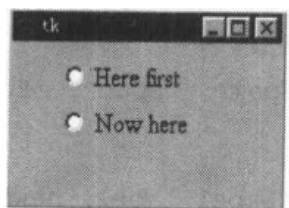


图 4-17 Radiobutton 窗口小部件

例子:

```
root=Tk()
ThisVariable=IntVar()
Radiobutton(root, text="Here first", variable=ThisVariable, value=1).pack()
Radiobutton(root, text="Now here", variable=ThisVariable, value=2).pack()
root.mainloop()
```

4.1.20 Scale (标尺) 窗口小部件

建立具有数值范围的滑块窗口小部件。用户上或下滑动它来返回一个数值，如图 4-18 所示。

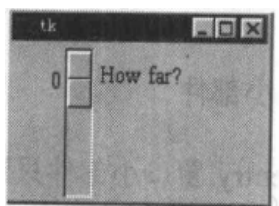


图 4-18 scale 窗口小部件

例子:

```
root=Tk()
scale=Scale(root, label="How far?")
scale.pack()
root.mainloop()
```

4.1.21 Scrollbar (滚动条) 窗口小部件

在框架或窗口旁边放置滚动条，允许用户向前或向后移动以完全显示文件。

例子:

```
root=Tk()
scrollbar=Scrollbar()
scrollbar.pack()
root.mainloop()
```

4.1.22 Text（文本）窗口小部件

创建文本输入区，看起来像一个文字处理器，如图 4-19 所示。没有默认的滚动条；你必须亲自添加一个。

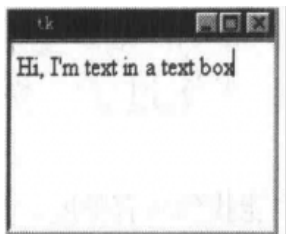


图 4-19 Text 窗口小部件

例子：

```
root=Tk()
text=Text(root)
text.pack()
root.mainloop()
```

4.1.23 Toplevel（顶层）窗口小部件

创建另一个窗口，作为根窗口的子窗口。

例子：

```
root=Tk()
toplevel=Toplevel(root)
root.mainloop()
```

第二部分 脚本

我刚刚在商店里逛过了，然后我要……

第2部分是一个仓库，在这里你能找到所有长度、形状和有用的脚本。这些脚本中有一些是我写的，另一个是由仁慈的、有天才的和超智能的程序员们提供的，不要忘记他们。

第5章 运行 Python

第6章 Tkinter 脚本

第7章 数据库

第8章 数学/科学

第9章 服务器

第10章 字符串和其他数据类型

第11章 系统操作和编程

第12章 游戏程序和人工智能

第5章 运行 Python

本章要点：

- GUI 模式下的主群组
- 主群组模式
- 交互模式

Python 有 4 种操作模式：

- GUI 下的主群组
- 主群组
- 嵌入
- 交互

当 Python 嵌入在另一种应用程序中时，你就会得到嵌入模式。这种模式适用于正式的编程需要并且不包含在本书中。查看 Python 发行版本文档。

5.1 GUI 模式下的主群组

GUI 模式下的主群组需要不同的编程风格，因为 GUI 需要事件循环，而不是典型脚本中的比较直接的风格。Tkinter 是和标准发行版本在一起的 Python 模块，拥有你对大多数 Python GUI 应用程序所需要的东西。第 6 章有 Tkinter GUI 脚本，所以你会有机会练习这种模式。

5.2 主群组模式

在主群组模式中，通过直接启动脚本来启动 Python，就好像脚本是可执行的一样。然而，某些系统首先需要一些配置。例如，在 Caldera Linux 中，从命令行运行 viking.py，你可以键入下面这些命令中的任何一个：

```
$ python viking.py 2
```

```
$ viking.py 2
```

在上面的每种情况下，2 都是传递到脚本的参数。

然而，在许多系统上（包括 Linux/GNU），除了下面的情况外没有一个会正常运行：

- 脚本在这条路径的目录中或在你当前存在的目录中；两地方都存在
- 执行为脚本设置的权限

不是每个系统都需要你设置权限和定义路径，但多数系统需要这样做。

为挽回错误，如果你能改变路径环境变量，现在立刻这样做，例如，在存储脚本的地方添加目录。在我的 Linux/GNU 机器上，默认目录是 /usr/lib/python1.5。换句话说，我只是把脚本放在同模块一样的目录中。这样很方便而且我没有混淆。然而，你也可以为开发的脚本创建一个新的目录，只要你把目录放进路径变量中。如果你不熟悉 Linux/GNU，查看你的文档看看怎样改变路径变量。

你将会看见许多在顶端具有特定行的 Python 脚本，它看起来像下面这个样子：

```
#!/usr/bin/python
```

这一行称为“pound-bang-hack”。在某些系统上，特别是 UNIX 系统，在主群组模式中 pound-bang-hack 告诉系统引起它的注意脚本是在什么时候在什么地方运行。你的系统也许需要也许不需要这一行，但是对于接触和其他人共享的脚本是有好处的。某些系统读取文件的第一行并得到指示，并且在那些系统上，这一行容易告诉系统在哪里能找到解释程序。幸运的是，因为 Python 解释程序不会关心任何以 # 开始的行，所以 Python 忽略了这一行。

如果你想要像主程序一样运行脚本，那么脚本需要一个 __main__ 例程序。当函数定义在顶端进行时，这个例程序按照惯例放在脚本的底部。一个典型的 __main__ 如下所示：

```
if __name__ == "__main__":
```

```
sheep_sighting= raw_input("Did you see a sheep in a tree? Answer Y or N:")
sheep_fly(sheep_sighting)
else:
    print "Module shpsite imported."
    print "To run, type: sheep_fly(*Y or N*)"
```

这些行检测一个变量，来查看脚本已注册的名字是否为“__main__”，如果你从命令行启动脚本，脚本的名字就会是“__main__”。在这种情况下，用户会得到提示，“Did you see a sheep in a tree? Answer Y or N:”。

另一方面，在交互模式中，如果脚本输入解释程序，用户会得到这样的消息，“Module shpsite imported. To run, type: sheep_fly()”。

如果是模块而不是主程序的脚本，就不需要这个代码。有些脚本倾向于其他模式，就需要这样的代码行。

5.3 交互模式

用于开发的有用模式是交互模式，这是四种模式中的最后一种。在这种模式下，Python 解释程序实时可用。在许多系统上你可以通过在命令行提示符下键入 `python` 命令来启动交互式会话。在 Windows 系统上，可以通过双击 Python 图标运行 Python。在 Linux/GNU 系统上，只需要键入单词 `python`。当 Python 交互式会话处于激活时，可以得到几行文本，后跟 Python 提示符，如下所示：

```
>>>
```

在交互模式下，你能做适用于只用一次函数的简单编程。例如，进行基本运算的这段代码：

```
>>>5*4
```

```
20
```

```
>>>
```

在这种情况下，解释程序不能保留任何事。操作结束后，如果你想重复操作，必须重新键入所有的东西。然而，你可以定义一个函数，它将保留在名字空间直到你取消会话。

```
>>>def run(var1, var2)
```

```
...     print var1*var2
```

```
...
```

```
>>>
```

当你看见第二行句点 (...) 时，你必须按 `Enter` 键以使解释程序知道正在用定义进行工作。要运行这个脚本，你可以键入：

```
>>>run(3,4)
```

同时 Python 将忠诚地响应：

```
12
```

```
>>>
```

你想让这项操作做多少次你就能让它做多少次。当你离开这个会话时，`run()`就会进入位存储桶并且丢失。

在交互模式下，你能把模块和脚本输入名字空间，这对于当你正在编写还没有为主群组模式做好准备的脚本时是很有帮助的。例如，如果你正在用 `get_instance()` 函数编写脚本 `covar.py`，你可以键入下面这些行来运行函数：

```
>>>import covar.py
```

```
>>>covar.get_instance()
```

`get_instance()`能做它被编写去做的无论什么事（有希望地）。

当解释程序输入模块或脚本时（模块通常是生活中具有不同目的脚本），解释程序检查语法错误。如果它发现一个语法错误，它就会让你立刻知道。直到你运行这段代码时运行错误才会显示。

本书中许多脚本都是在主群组模式下编写和使用的，虽然如果你想让它们在交互模式下运行时它们也能运行。

不同系统退出 Python 的方法不同。要使 Python 离开，需要一个 EOF 字符，在 Linux 系统上是 `Control+D`。在 Linux 系统上使用 `Control+Z` 会使解释程序停止，但不能退出它。

第 6 章 Tkinter 脚本

本章要点:

- Andy 的窗口小部件
- Regexer.py
- dictEdit.py
- engine.py

为了补偿我们中那些除了代码之外还想要 GUI (图形用户界面) 的人, 标准 Python 发行版本使用 Tk。你几乎能使用你想使用的任何图形工具包, 和许多 Python 程序员另外特别喜欢的工具。Tkinter 是和 Tk 调用连接的 Python 模块。

唯恐任何人认为 Python 是一种小型语言, 不能作为一种应用程序语言, 查看 Grail, 一个完全用 Python 构建的浏览器。Grail 使用 Tkinter, 并且在附带 CD 上可以得到。

6.1 Andy 的窗口小部件

```
""" FWidgets 4.2 版本, 时间: <96/12/12 12:36:05 x-aes>
```

一些我经常使用的简单窗口构架, 决定存放在这里:

```
Copyright (C) 1996 Andy Eskilsson, flognat@fukt.hk-r.se
```

这是一个免费软件, 可以根据 GPL 的条款进行不受限的传播。有关该软件的授权条件的细节, 请参阅 GNU General Public License (GUN 通用公众授权)。

下面是文档说明:

OkCancelWindow:

一个简单的窗口, 它具有两个按钮、文本和标签, 还可以配置按钮。调用就会弹出。

返回 TRUE/FALSE

ScrolledText

一个带有文本框和滚动条的框架。初始程序带有 Master、textconfig 和 scrollbarconfig。set 将清除对话框并且输入的字符串填充它, 而 get 则将文本作为字符串返回。

LabelEntry

包含一个文本标签和一个数据入口字段。初始程序带有 Master、Text、labelconfig 和 textconfig。empty、get、set 分别清除、获取、设置入口, 并将事件绑定到回调函数上。

OKCancelButtons

一个包含两个按钮的框架。初始程序带有 master、okbutton 配置和 cancelbutton 配置，以及作为配置字典中的“指令”的回调函数的 send。

ScrolledList

一个包含列表框和滚动条的框架。

初始程序带有 Master、listconfiguration 和 scrollbarconfiguration。bind 用于将事件绑定到回调函数上，empty 清空列表，add 添加一个字符串或向框中添加一条或一串字符串，默认情况下，结束时 active 返回最后选中的字符串，get 返回选中的字符串或字符串列表，getIndex 返回选中条目的#，unselect 消除所有的选择。

要与我（作者）联系，请给 flognat@fukt.hk-r.se 发信。可以在 <http://www.fukt.hk-r.se/~flognat> 上找到本文件的最新版本。

如果你喜欢，发明信片给 Andy Eskilsson

Kämnärsv.. 3b228

S-226 46 Lund

Sweden

"""

```
from Tkinter import *
from types import *
from string import *
```

```
defu DummyCommand():
    pass
```

```
scrollbarWidth = 10
```

```
class OkCancelWindow(Frame):
    def __init__(self, Master, Text, Label='Are U sure?', ytcnf={}, ntcnf={}):
        Frame.__init__(self, Master)
        self.pack()
        self._top=Toplevel(Master)
        self._top.title(Label)
        ytcnf['command']=self.okBtn
        ntcnf['command']=self.noBtn
        self.createWindow(self._top, Text, ytcnf, ntcnf)
```

```
def go(self):
    self._top.grab_set()
    try:
        self._top.mainloop()
    except SystemExit, selection:
        self._top.destroy()
    return selection

def createWindow(self, Master, Text, ytcnf, ntcnf):
    self._text=Label(Master, {'text' : Text})
    self._text.pack({'side': 'top', 'fill' : 'x'})
    self._buttons=OkCancelButtons(Master, ytcnf, ntcnf)
    self._buttons.pack({'side': 'bottom', 'fill' : 'x'})

def okBtn(self):
    raise SystemExit, TRUE

def noBtn(self):
    raise SystemExit, FALSE

class ScrolledText(Frame):
    def __init__(self, Master, txtcnf={}, sbcnf={}):
        Frame.__init__(self, Master)
        self.pack()

        self._textArea=Text(self, txtcnf)
        self._textArea.pack({'side': 'left', 'fill': 'both', 'expand' : 'yes'})
        if not sbcnf.has_key('width'):
            sbcnf['width']=scrollbarWidth
        self._scrollBar=Scrollbar(self, sbcnf)
        self._scrollBar.pack({'side': 'right', 'fill': 'y', 'expand' : 'yes'})
        self._textArea['yscrollcommand'] = (self._scrollBar, 'set')
        self._scrollBar['command'] = (self._textArea, 'yview')

    def set(self, text):
```

```
self._textArea.delete("1.0", END)
self._textArea.insert(END, text)

def get(self):
    return self._textArea.get("1.0", END)

class LabelEntry(Frame):
    def __init__(self, Master, Text, lblcnf={}, txtcnf={}):
        Frame.__init__(self, Master)
        self.pack()
        lblcnf['text'] = Text
        self._label = Label(self, lblcnf)
        self._label.pack({'side': 'left', 'expand': 'no'})

        self._entry = Entry(self, txtcnf)
        self._entry.pack({'side': 'right', 'expand' : 'yes', 'fill' : 'x'})

    def empty(self):
        self._entry.delete(0, END)

    def get(self):
        return self._entry.get()

    def set(self, Value=''):
        self.empty()
        self._entry.insert(END, Value)

    def bind(self, Event, Callback):
        self._entry.bind(Event, Callback)

class OkCancelButtons(Frame):
    def __init__(self, Master, okcnf = {}, cancelcnf = {}):
        Frame.__init__(self, Master)
        self.pack()
```

```

        if not okcnf.has_key('text'):
            okcnf['text']='Ok'
        if not cancelcnf.has_key('text'):
            cancelcnf['text']='Cancel'

        if not okcnf.has_key('command'):
            okcnf['command']=DummyCommand
        if not cancelcnf.has_key('command'):
            cancelcnf['command']=DummyCommand

        self._cancelBtn = Button(self, cancelcnf)
        self._cancelBtn.pack({'side': 'right'})

        self._okBtn = Button(self, okcnf)
        self._okBtn.pack({'side': 'right'})

class ScrolledList(Frame):
    def __init__(self, Master, listcnf={}, sbcnf={}):
        Frame.__init__(self, Master)
        self.pack()

        if not sbcnf.has_key('width'):
            sbcnf['width']=scrollbarWidth

        self._multiselect=False
        if listcnf.has_key('selectmode'):
            if listcnf['selectmode'] == 'multiple' or listcnf['selectmode'] ==
'extended':
                self._multiselect=True

        self._list = Listbox(self, listcnf)
        self._list.pack({'side': 'left', 'fill': 'both', 'expand' : 'yes'})

        self._sb = Scrollbar(self, sbcnf)
        self._sb.pack({'side': 'right', 'fill': 'y'})

```

```
self._list['yscrollcommand'] = (self._sb, 'set')
self._sb['command'] = (self._list, 'yview')

def bind(self, Event, Callback):
    self._list.bind(Event, Callback)

def empty(self, begin=0, end=END):
    self._list.delete(0, END)

def add(self, item, position = END):
    if type(item) == StringType:
        self._list.insert(position, item)
    elif type(item) == ListType:
        for i in item:
            self._list.insert(position, i)

# 返回激活的项目
def active(self):
    return self._list.get(ACTIVE)

def whatis(self, first, last=None):
    if last==None:
        return self._list.get(first)
    else:
        # 黑客 Galore, 为什么这没有在 Tkinter 中实现??
        return self._list.tk.splitlist(self._list.tk.call(self._list._w, 'get',
first, last))

# 返回所选条目或具有所选条目的列表
def get(self):
    if self._multiselect==FALSE:
        retVal=self._list.curselection()
        if len(retVal)==0:
            return None
```

```
    else:
        return self._list.get(retVal[0])
    else:
        ret=[]
        for i in self._list.curselection():
            ret.append(self._list.get(i))
        return ret

# 返回所选条目的#
def getIndex(self):
    selection=self._list.curselection()
    if len(selection)==0:
        return None
    if self._multiselect==FALSE:
        return atoi(selection[0])
    else:
        retVal=()
        for i in selection:
            retVal.append(atoi(i))
        return retVal
```

通常可以只有列表和一条字符串，不过如果在同名列表中拥有一条以上的条目时，就会发生有
趣的事情。

```
def select(self, items):
    noOfItems=self._list.size()
    if type(items) == StringType:
        for i in range(noOfItems):
            if items==self._list.get(i):
                self._list.select_set(i,i)
    elif type(items) == ListType:
        for i in range(noOfItems):
            line=self._list.get(i)
            for j in items:
                if line==j:
                    self._list.select_set(i,i)
```

```

        else:
            pass # 可以产生出一个异常, 但是:-)

    def unselect(self):
        self._list.select_clear(0)

##### TEST #####

root=None
GoodBye='bye bye'

def testfun():
    print root.slist.get(),
    print root.slist.getIndex()

def testfun2():
    a=OkCancelWindow(root, 'OOPs really, really cancel?')
    b=a.go()
    print b
    if b:
        raise GoodBye

def test():
    root.slist = ScrolledList(root)#, {'selectmode' : 'multiple'})
    root.entry = LabelEntry(root, 'Enter')
    root.btn = OkCancelButtons(root, {'command': testfun}, {'command':
testfun2})
    root.txt = ScrolledText(root)
    root.txt.set("abc123")
    root.txt2 = ScrolledText(root, {'relief': 'sunken', 'width': 10}, {'width':
3})

    root.slist.add(["A", "B", "C", "D", "E", "F", "G"])
    root.slist.select("B")

    try:

```

```

        root.mainloop()
    except GoodBye:
        root.destroy()

if __name__ == '__main__':
    root = Tk()
    test()

```

它怎样工作

这个代码是本章伟大的开始，因为它是一些基本的和方便的窗口小部件的编译集合。这些窗口小部件包括：

- OkCancelWindow，具有两个按钮、文本和标签的简单窗口
- ScrolledText，具有文本框和滚动条的框架
- LabelEntry，包含文本标签和数据的输入字段的框架
- OKCancelButton，包含两个按钮的框架
- ScrolledList，包含列表框和滚动条的框架

正如你所期望的，窗口小部件被组合进类中。你可以按现状使用这些类或扩展它们。你只在这出现的窗口小部件就可以在 Tkinter 做大量工作。

由 Andrew Markebo（在 Telelogic AB 的软件开发人员）提供的，“可用性位于用户的心目中...”。

在脚本中函数被调用之前必须先定义。这并不意味着它们在调用之上必须被物理地定义，而是必须及时地比调用较早地定义。

6.2 Regexer.py

Python 有两个使用正规表达式标准模块：regex 和 re。regex 是两个中比较旧的，而且有很少的功能。这两个模块经常混淆，因为许多人从来没有使用过正规表达式。这个脚本使正规表达式看起来很直观。

```

""" Regexer 3.0 版，时间: <96/11/03 22:53:54 flognat>
一个简单的正规表达式模拟器

Copyright (C) 1996 Andy Eskilsson

```

本软件提供给大家，不过既无书面，也无隐含的担保。任何个人或组织，无论出于何种目的，均可免费使用、拷贝、修改、发行或销售该软件，不过要在所有的拷贝中出现上述版权声明和本段文字。

下面是文档说明。

我是唯一一个发现\\(.*)可读性很差的人吗？想要试着弄清怎样清除错误吗？这就是 Regexer 诞生的理由！

Regexer 是一个在 Python 中进行语法检查的小应用程序。它能立即给予用户反馈，告诉对 Regexer 进行什么修改，或者 Regexer 作用其上的字符串对于 Regexer 意味什么。

我想试着学习 Regexer 时它也可以在手边，因为可能试验它们。

regexps/strings 可以保存到分隔的列表框中以便在不同的 regexps/strings 间快速切换。并且你可以通过滑动幻灯片到不同的组中。

在各自的窗口中选择 regexps/strings（用鼠标右键），分别发送到主 regexer 窗口中的域中，在各自的窗口中单击左键删除选定的入口。不要单击入口（除非没有选定）。如果有人有更多的办法做这件事，非常欢迎。

我考虑的是 egrep 版本问题，给它一个字符串分支，它列出相匹配的一个，模拟的可能性一次超过一个。如果有人觉得他们喜欢这些特色，我或许可以实现它们。

要与我联系，请发信给 flognat@fukt.hk-r.se，可以在 <http://www.fukt.hk-r.se/~flognat> 上找到该文件的最新版本。

如果你喜欢（而又不必依赖它，收信总是十分有趣的），请发明信片至：Andy Eskilsson

Kåmnärsv.3b228

S-226 46 Lund

Sweden

"""

' <--- 为使字体锁定模式或更为好看所需要的 :-)

from Tkinter import *

from StringIO import StringIO

import regex

import regex_syntax

import string

#####

#只是一些工具类、应用程序还在下面

#####

def bittify(value, bits):

count=1

```
retlist=[]

for bit in range(bits):
    if count&value:
        retlist.append(1)
    else:
        retlist.append(0)
        count=count*2

return retlist

def unbittify(bits):
    count=1
    retval=0

    for bit in bits:
        if bit:
            retval=retval^count
            count=count*2

    return retval

def doSearch(cregexp, str, fp):
    try:
        s=cregexp.search(str)
    except regex.error, message:
        fp.write("Error in search:\n %s" % message)
    else:
        fp.write("search returns %d\n" % s)
        if s!=-1:
            fp.write(" and points on '%s'\n\n" % string.strip(str[s:s+15]))

def doMatch(cregexp, str, fp):
    try:
        m=cregexp.match(str)
```

```
except regex.error, message:
    fp.write("Error in match:\n %s" % message)
else:
    fp.write("match returns %d\n\n" % m)

def doGroups(cregexp, str, groups, fp):
    try:
        strgroups=apply(cregexp.group, tuple(range(1, groups+1)))
    except regex.error, message:
        fp.write("Error in groups:\n %s" % message)
    else:
        if groups==1:
            fp.write("And the %d group is:\n" % groups)
            fp.write(" %s\n" % strgroups)
        else:
            fp.write("And the %d groups are:\n" % groups)
            for group in strgroups:
                fp.write(" %s\n" % group)

class ListWindow(Frame):
    """ 一个具有滚动列表的窗口 """
    def __init__(self, title, master=None):
        """ title 是窗口的标题, master 是窗口最后的主控 """
        Frame.__init__(self, master)
        self._top=Toplevel(self.master)
        self._top.title(title)
        self.pack()

        self._multiselect=FALSE
        self.drawme()

    def drawme(self):
        self._list=Listbox(self._top)
        self._list.pack({'side': 'left', 'fill': 'both', 'expand' : 'yes'})
        self._sb = Scrollbar(self._top)
```

```
self._sb.pack({'side': 'right', 'fill': 'y'})

self._list['yscrollcommand'] = (self._sb, 'set')
self._sb['command'] = (self._list, 'yview')

def bind(self, Event, Callback):
    """ 将列表事件绑定到 Callback """
    self._list.bind(Event, Callback)

def clear(self):
    """ 清除列表 """
    self._list.delete(0, END)

def add(self, item, position = END):
    """ 向列表添加一个或更多个条目 """
    if type(item) == StringType:
        self._list.insert(position, item)
    elif type(item) == ListType:
        for i in item:
            self._list.insert(position, i)

def active(self):
    """ 返回活动的条目 """
    return self._list.get(ACTIVE)

def whatis(self, first, last=None):
    """ 返回 item[s] """
    if last==None:
        return self._list.get(first)
    else:
        # 黑客 Galore, 为什么这没有在 Tkinter 中实现??
        return self._list.tk.splitlist(self._list.tk.call(self._list._w, 'get',
first, last))

def get(self):
```

```
""" 返回一选定条目或选定条目的列表 """
if self._multiselect==FALSE:
    retVal=self._list.curselection()
    if len(retVal)==0:
        return None
    else:
        return self._list.get(retVal[0])
else:
    ret=[]
    for i in self._list.curselection():
        ret.append(self._list.get(i))
    return ret

# 返回选定条目的#
def getIndex(self):
    selection=self._list.curselection()
    if len(selection)==0:
        return None
    if self._multiselect==FALSE:
        return atoi(selection[0])
    else:
        retVal=()
        for i in selection:
            retVal.append(atoi(i))
        return retVal

# 通常我们可以只有列表和一个字符串，不过在同名列表中有一个以上的条目时会发生有趣的事情
def select(self, items):
    noOfItems=self._list.size()
    if type(items) == StringType:
        for i in range(noOfItems):
            if items==self._list.get(i):
                self._list.select_set(i,i)
    elif type(items) == ListType:
        for i in range(noOfItems):
```

```

        line=self._list.get(i)
        for j in items:
            if line==j:
                self._list.select_set(i,i)
        else:
            pass #可以设置一个异常, 不过:- )

def delete(self, item):
    self._list.delete(item)

def unselect(self):
    self._list.select_clear(0)

#####
# 在这可以开始真正的应用程序
#####

class Regexer(Frame):
    """应用程序本身"""
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self._matchCheck = BooleanVar()
        self._searchCheck = BooleanVar()
        self._linesCheck = BooleanVar()

        self._RE_NO_BK_PARENS = BooleanVar()
        self._RE_NO_BK_VBAR = BooleanVar()
        self._RE_BK_PLUS_QM = BooleanVar()
        self._RE_TIGHT_VBAR = BooleanVar()
        self._RE_NEWLINE_OR = BooleanVar()
        self._RE_CONTEXT_INDEP_OPS = BooleanVar()

        self.master.title('Regexer')
        self.pack()
        self.drawme()

```

```
def calculate(self, event=None):
    """ 发送结果, 取回报告。"""

    reg=self._regexinputframe.entry.get()
    if reg=='':
        return

    self._textoutput.delete("1.0", END)

    regex.set_syntax(unbittify((self._RE_NO_BK_PARENS.get(),
                                self._RE_NO_BK_VBAR.get(),
                                self._RE_BK_PLUS_QM.get(),
                                self._RE_TIGHT_VBAR.get(),
                                self._RE_NEWLINE_OR.get(),
                                self._RE_CONTEXT_INDEP_OPS.get()))))

    try:
        cregex=regex.compile(reg)
    except regex.error, message:
        self._textoutput.insert(END, "Error in compilation of regex:\n %s"
                                % message)
    return

    fp=StringIO()
    str=self._stringinputframe.entry.get("1.0", END)

    if self._searchCheck.get():
        doSearch(cregex, str, fp)

    if self._matchCheck.get():
        doMatch(cregex, str, fp)

    if self._groups.get():
        doGroups(cregex, str, self._groups.get(), fp)
```

```
self._textoutput.insert(END, fp.getvalue())
fp.close()

def stringschanged(self, event=None):
    self._stringswindow.unselect()
    self.calculate()

def regexpschanged(self, event=None):
    self._regexswindow.unselect()
    self.calculate()

def savestring(self, event=None):
    str=self._stringinputframe.entry.get("1.0", END)
    #我不想保存空字符串!
    if string.strip(str)!='':
        self._stringswindow.add(str)

def saveregexp(self, event=None):
    rexp=self._regexinputframe.entry.get()
    # 我不想保存空字符串!
    if rexp!='':
        self._regexswindow.add(rexp)

def grabregexp(self, event=None):
    self._regexinputframe.entry.delete(0, END)
    rexp=self._regexswindow.get()
    if rexp!=None:
        self._regexinputframe.entry.insert(END, rexp)

def grabstring(self, event=None):
    self._stringinputframe.entry.delete("1.0", END)
    str=self._stringswindow.get()
    if str!=None and string.strip(str)!='':
        self._stringinputframe.entry.insert(END, str)
```



```
def stringdelete(self, event=None):
    self._stringswindow.delete(ACTIVE)

def rexpdelete(self, event=None):
    self._regexswindow.delete(ACTIVE)

def setsyntax(self, value):
    bits=bittify(value, 6)

    self._RE_NO_BK_PARENS.set(bits[0])
    self._RE_NO_BK_VBAR.set(bits[1])
    self._RE_BK_PLUS_QM.set(bits[2])
    self._RE_TIGHT_VBAR.set(bits[3])
    self._RE_NEWLINE_OR.set(bits[4])
    self._RE_CONTEXT_INDEP_OPS.set(bits[5])

def AWKSyntax(self, event=None):
    self.setsyntax(regex_syntax.RE_SYNTAX_AWK)

def ECREPSyntax(self, event=None):
    self.setsyntax(regex_syntax.RE_SYNTAX_EGREP)

def GREPSyntax(self, event=None):
    self.setsyntax(regex_syntax.RE_SYNTAX_GREP)

def EMACSSyntax(self, event=None):
    self.setsyntax(regex_syntax.RE_SYNTAX_EMACS)

def drawme(self):
    self._inputframe=Frame(self)

    self._regexinputframe=Frame(self._inputframe)
    self._regexinputframe.text=Label(self._regexinputframe, {'text' :
        'Regexp: '})
```

```

self._regexinputframe.text.pack({'side': 'left', 'expand': 'no'})
self._regexinputframe.entry=Entry(self._regexinputframe)
self._regexinputframe.entry.bind('<KeyRelease>',self.regexpschanged)
self._regexinputframe.entry.pack({'side': 'right', 'expand': 'yes',
    'fill' : 'x'})
self._regexinputframe.pack({'side' : 'top', 'expand' : 'yes', 'fill' :
    'x'})

self._stringinputframe=Frame(self._inputframe)
self._stringinputframe.text=Label(self._stringinputframe, {'text' :
    'String:'})
self._stringinputframe.text.pack({'side': 'left', 'expand': 'no'})
self._stringinputframe.entry=Text(self._stringinputframe, {'height' :
    '3', 'width' : 30})
self._stringinputframe.entry.bind('<KeyRelease>',self.stringschanged)
self._stringinputframe.entry.pack({'side': 'right', 'expand': 'yes',
    'fill' : 'x'})
self._stringinputframe.pack({'side' : 'bottom', 'expand' : 'yes',
    'fill' : 'x'})

self._inputframe.pack({'expand' : 'yes', 'fill' : 'x'})

self._controlframe=Frame(self)

self._groups=Scale(self._controlframe, {'from' : 0, 'to' : 10,
    'orient' : 'horiz',
    'label' : 'groups',
    'command' : self.calculate })
self._groups.pack({'side' : 'left'})

self._buttonframe=Frame(self._controlframe)
self._buttonframe.pack({'side' : 'left'})

self._saveregex=Button(self._buttonframe, {'text' : 'Save regexp'})

```

```
self._saverexgex.bind('<ButtonRelease-1>', self.saverexgexp)
self._saverexgex.pack(side='top', fill='x', expand='yes')

self._savestring=Button(self._buttonframe, {'text' : 'Save string'})
self._savestring.bind('<ButtonRelease-1>', self.savestring)
self._savestring.pack(side='top', fill='x', expand='yes')

self._syntaxmenubar=Menubutton(self._buttonframe, text='Syntaxes',
                                relief=RAISED)

syntaxmenu = Menu(self._syntaxmenubar)
syntaxmenu.add_checkbutton(label='RE_NO_BK_PARENS',
                           variable=self._RE_NO_BK_PARENS,
                           command=self.calculate)
syntaxmenu.add_checkbutton(label='RE_NO_BK_VBAR',
                           variable=self._RE_NO_BK_VBAR,
                           command=self.calculate)
syntaxmenu.add_checkbutton(label='RE_BK_PLUS_QM',
                           variable=self._RE_BK_PLUS_QM,
                           command=self.calculate)
syntaxmenu.add_checkbutton(label='RE_TIGHT_VBAR',
                           variable=self._RE_TIGHT_VBAR,
                           command=self.calculate)
syntaxmenu.add_checkbutton(label='RE_NEWLINE_OR',
                           variable=self._RE_NEWLINE_OR,
                           command=self.calculate)
syntaxmenu.add_checkbutton(label='RE_CONTEXT_INDEP_OPS',
                           variable=self._RE_CONTEXT_INDEP_OPS,
                           command=self.calculate)
syntaxmenu.add_separator()
syntaxmenu.add_command(label='RE_SYNTAX_AWK', command=self.AWKSyntax)
syntaxmenu.add_command(label='RE_SYNTAX_EGREP',
                        command=self.EGREPSyntax)
syntaxmenu.add_command(label='RE_SYNTAX_GREP',
                        command=self.GREPSyntax)
```

```

syntaxmenu.add_command(label= RE_SYNTAX Emacs',
                        command=self.EMACSSyntax)
self._syntaxmenubar[ menu'] = syntaxmenu
self._syntaxmenubar.pack(fill='x', expand='yes')

# Button(self._buttonframe, {'text' : 'Save regexp'})

self._checkframe2=Frame(self._controlframe)
self._othercheck=Checkbutton(self._checkframe2, text="Other",
                              anchor=W, onvalue=1, offvalue=0)
self._othercheck.pack()

self._checkframe2.pack({'side' : 'right'})

self._checkframe=Frame(self._controlframe)
self._matchCheckb=Checkbutton(self._checkframe, text="Match", anchor=W,
                              command=self.calculate,
                              variable=self._matchCheck)
self._matchCheckb.pack(expand= YES', fill='x')
self._searchCheckb=Checkbutton(self._checkframe, text="Search", anchor=W,
                              command=self.calculate,
                              variable=self._searchCheck)
self._searchCheckb.pack(expand= YES', fill='x')
self._linesCheckb=Checkbutton(self._checkframe, text="Selected lines",
                              command=self.calculate,
                              variable=self._linesCheck)
self._linesCheckb.pack(expand= YES', fill= x')
self._checkframe.pack({'side' : 'right'})

self._controlframe.pack()

self._outputarea=Frame(self)
self._textoutput=Text(self._outputarea, width=55)
self._textoutput.pack()
self._outputarea.pack(side='bottom', expand='yes', fill='both')

```

```

self.pack()

self._stringswindow=ListWindow( Strings', self)
self._stringswindow.bind( <ButtonRelease-1>', self.grabstring)
self._stringswindow.bind('<ButtonRelease-3>', self.stringdelete)

self._regexwindow=ListWindow('Regexps', self)
self._regexwindow.bind('<ButtonRelease-1>' , self.grabregexp)
self._regexwindow.bind('<ButtonRelease-3>' , self.rexpdelete)

if __name__=="__main__":
    a=Regexer()
    a.mainloop()

```

它怎样工作

你能从提供者的 Web 网站 www.fuict.hek-r.se/~flognat/hacks 得到一些关于这个脚本的有趣的信息。

这个脚本只使用两个 GUI 类：Regexer 和 ListWindow。主程序比较简单，只有两行，第二行 `a.mainloop()`，调用 Regexer 的 `mainloop()` 方法，使脚本等待事件。这是 GUI 编程所必需的。

除了 Tkinter 代码以外，脚本正在进行基本 regex 字符串搜索。参见 Python 发行版本文档，以得到关于 regex 和正则表达式操作的详细资料。

由 Andrew Markebo (Telelogic AB 的软件开发人员) 提供的。“实用性位于用户的心目中...”

6.3 dictEdit.py

这个脚本对于数据库的 GUI 前端是很有用的基础。它编辑字典类型对象，包括结构，例如 shelves。你可以添加项目或查看它们，但在这个实现中你不能删除它们。从命令行运行 `dictEdit.py`，来查看一个生动的测试。

```

"""

```

用户 x 接口和一个小的类定义来编辑字典/数据库/框架。

检查下面的测试以获取更多的信息！

```

        flognat@fukt.hk-r.se

"""

import string

from Tkinter import *

#import FWidgets

class ScrolledList(Frame):
    def __init__(self, Master, listcnf={}, sbcnf={}):
        Frame.__init__(self, Master)
        #我们应该让上面的“家伙”做这个吗?
        self.pack(fill=BOTH, expand=YES)

#     if not sbcnf.has_key('width'):
#         sbcnf['width']=scrollbarWidth

        self._multiselect=FALSE
        if listcnf.has_key('selectmode'):
            if listcnf['selectmode'] == 'multiple' or listcnf['selectmode'] ==
                'extended': self._multiselect=TRUE

        self._list = Listbox(self,listcnf)
        self._list.pack({'side': 'left', 'fill': 'both','expand' : 'yes'})

        self._sb = Scrollbar(self, sbcnf)
        self._sb.pack({'side': 'right', 'fill': 'y'})

        self._list['yscrollcommand'] = (self._sb, 'set')
        self._sb['command'] = (self._list, 'yview')

    def bind(self, Event, Callback):
        self._list.bind(Event, Callback)

    def empty(self, begin=0, end=END):

```

```
self._list.delete(0, END)

def add(self, item, position = END):
    if type(item) == StringType:
        self._list.insert(position, item)
    elif type(item) == ListType:
        for i in item:
            self._list.insert(position, i)

#返回活动条目
def active(self):
    return self._list.get(ACTIVE)

def whatis(self, first, last=None):
    if last==None:
        return self._list.get(first)
    else:
        # Hack Galore, 为什么这没有在 Tkinter 中实现呢??
        return self._list.tk.splitlist(self._list.tk.call(self._list._w, 'get',
            first, last))

# 返回所选条目或具有所选条目的列表
def get(self):
    if self._multiselect==FALSE:
        retVal=self._list.curselection()
        if len(retVal)==0:
            return None
        else:
            return self._list.get(retVal[0])
    else:
        ret=[]
        for i in self._list.curselection():
            ret.append(self._list.get(i))
        return ret

# 返回所选条目的#
def getIndex(self):
```

```

selection=self._list.curselection()
if len(selection)==0:
    return None
if self._multiselect==FALSE:
    return atoi(selection[0])
else:
    retVal=()
    for i in selection:
        retVal.append(atoi(i))
    return retVal

```

#通常我们可以只有列表和一个字符串，但是如果在同名列表中有一条以上的条目时会发生有趣的事情

```

def select(self, items):
    noOfItems=self._list.size()
    if type(items) == StringType:
        for i in range(noOfItems):
            if items==self._list.get(i):
                self._list.select_set(i,i)
    elif type(items) == ListType:
        for i in range(noOfItems):
            line=self._list.get(i)
            for j in items:
                if line==j:
                    self._list.select_set(i,i)
    else:
        pass # 可以设置一个异常，但是:-)

```

```

def unselect(self):
    self._list.select_clear(0)

```

```

class LabelEntry(Frame):
    def __init__(self, Master, Text, lblcnf={}, txtcnf={}):
        Frame.__init__(self, Master)
        self.pack()
        lblcnf['text']= Text

```



```
self._label = Label(self, lblcnf)
self._label.pack({'side': 'left', 'expand': 'no'})

self._entry = Entry(self, txtcnf)
self._entry.pack({'side': 'right', 'expand' : 'yes', 'fill' : 'x'})

def empty(self):
    self._entry.delete(0, END)

def get(self):
    return self._entry.get()

def set(self, Value=''):
    self.empty()
    self._entry.insert(END, Value)

def bind(self, Event, Callback):
    self._entry.bind(Event, Callback)

class GetDialog(Toplevel):
    def __init__(self, parent, title = None, prompt='foo'):
        Toplevel.__init__(self, parent)
        self.transient(parent)
        self.value=None
        if title:
            self.title(title)

        self.parent = parent
        self.result = None
        self.prompt=prompt
        body = Frame(self)
        self.initial_focus = self.body(body)
        body.pack(padx=5, pady=5)
```

```
self.buttonbox()

self.grab_set()

if not self.initial_focus:
    self.initial_focus = self

self.protocol('WM_DELETE_WINDOW', self.cancel)

self.initial_focus.focus_set()
self.wait_window(self)

def body(self, master):
    w = Label(master, text=self.prompt, justify=LEFT)
    w.grid(row=0, padx=5, sticky=W)

    self.entry = Entry(master, name="entry")
    self.entry.grid(row=1, padx=5, sticky=W+E)

    return self.entry

def buttonbox(self):
    box = Frame(self)

    w = Button(box, text="OK", width=10, command=self.ok)
    w.pack(side=LEFT, padx=5, pady=5)
    w = Button(box, text="Cancel", width=10, command=self.cancel)
    w.pack(side=LEFT, padx=5, pady=5)

    self.bind("<Return>", self.ok)
    self.bind("<Escape>", self.cancel)

    box.pack()

def getresult(self):
```

```
        return self.value

    def ok(self, event=None):
        self.value=self.entry.get()
        self.withdraw()
        self.update_idletasks()
        self.cancel()

    def cancel(self, event=None):
        self.parent.focus_set()
        self.destroy()

def getValue(parent, title, prompt):
    foo=GetDialog(parent, title, prompt)
    return foo.getresult()

doMethod="XyZZy"

class Executor:
    def __init__(self, method, master):
        self._meth=method
        self._master=master

    def __call__(self):
        self._master.callmeth(self._meth)

class FormEditor(Frame):
    def __init__(self, Master=None, dict, fields):
        Frame.__init__(self, Master)
        self._form={}
        self._buttons={}
        self._dict=dict
        self._key=None
        self._drawme(fields)
        self.pack()
```

```

def _drawme(self, fields):
    formFrame=Frame(self)
    formFrame.pack(side='top', fill='both', expand='yes', anchor='n')
    for descr, name, type in fields:
        if type is doMethod:
            self._buttons[name]=Executor(name, self)
            Button(formFrame, text=descr,command=self._buttons[name],
                    foreground='blue').pack(side='bottom')
            continue

        self._form[name]=type()
        LabelEntry(formFrame, descr,
                    txtcnf={'textvariable' : self._form[name]},
                    lblcnf={'anchor' : 'w'}).pack(side='top',
                                                    fill='x')

    buttonframe=Frame(self)
    buttonframe.pack(side='bottom', anchor='n')
    savebutton=Button(buttonframe, text='Save', command=self.save)
    savebutton.pack(side='right')
    cancelbutton=Button(buttonframe, text='Recall', command=self.recall)
    cancelbutton.pack(side='right')
    btnframe=Frame(self)
    btnframe.pack(side='bottom')
    Button(btnframe, text='Copy', command=self.copy).pack(side='left')
    Button(btnframe, text='Rename', command=self.rename).pack(side='left')

def fill(self, key):
    self._key=key
    item=self._dict[key]

    for field in self._form.keys():
        self._form[field].set(item.__dict__[field])

```

```
def callmeth(self, meth):
    if not self._key:
        return

    item=self._dict[self._key]

    if hasattr(item, meth):
        getattr(item, meth)()
    else:
        print 'Function %s not found' % meth

    self._dict[self._key]=item
    self.fill(self._key)

def rename(self, event=None):
    newkey=getValue(self, "Renaming", "Enter new key")
    if newkey!=" " and newkey!=None:
        item=self._dict[self._key]
        del self._dict[self._key]
        self._key=newkey
        self._dict[newkey]=item

def copy(self, event=None):
    newkey=getValue(self, "Copying", "Enter new key")
    if newkey!=" " and newkey!=None:
        item=self._dict[self._key]
        self._dict[newkey]=item

def save(self, event=None):
    if not self._key:
        return

    item=self._dict[self._key]
    for field in self._form.keys():
        item.__dict__[field]=self._form[field].get()
```

```
        self._dict[self._key]=item

    def recall(self):
        self.fill(self._key)

class DICTEditor(Frame):
    def __init__(self, Master=None, dict, formdef):
        Frame.__init__(self, Master)
        self.pack(expand='yes', fill='both')

        self._formdef=formdef
        self._dict=dict
        self._oldstr=""
        self._drawme()
        self._keys=self._dict.keys()
        self._viewdata=dict.keys()
        self._viewcache=[]
        self._listbox.add(self._viewdata)

    def bye(self, event=None):
        self.quit()
        #保存表单中的数据

    def refresh(self):
        self._keys=self._dict.keys()
        self._newchar(None, 1)

    def _drawme(self):
        leftframe=Frame(self)
        leftframe.pack(side='left', expand='yes', fill='x', anchor='n')
        self.editor=FormEditor(leftframe, self._dict, self._formdef)
        self.editor.pack(side='top', expand='yes', fill='both')

        bframe=Frame(leftframe)
```

```
bframe.pack(side='bottom', anchor='s', expand='yes', fill='both')

Button(bframe, text='Refresh', command=self.refresh).pack(side='left',
                                                         anchor='s')
Button(bframe, text='Quit', command=self.bye,
       foreground='red').pack(side='left', anchor='s')

rightframe=Frame(self)
rightframe.pack(side='right', expand='yes', fill='both')

self._listbox=ScrolledList(rightframe)
self._listbox.bind('<ButtonRelease-1>', self._selected)
self._listbox.pack(side='top', expand='yes', fill='both')

self._entry=Entry(rightframe)
self._entry.bind('<KeyRelease>', self._newchar)
self._entry.pack(side='bottom', fill='x')

def _newchar(self, event=None, upd=None):
    str=self._entry.get()

    # 好的, 我们不需要担心更新
    if len(str) is 0:
        self._viewdata=self._keys
        self._viewcache=[]
    elif upd:
        newview=[]
        for key in self._viewdata:
            if string.find(key, str)!=-1:
                newview.append(key)
        self._viewdata=newview

    elif len(str)==len(self._oldstr):
        return

    elif len(str)>len(self._oldstr):
```

```

        self._viewcache.insert(0, self._viewdata)
        newview=[]
        for key in self._viewdata:
            if string.find(key, str)!=-1:
                newview.append(key)

        self._viewdata=newview
    else:
        self._viewdata=self._viewcache[0]
        del self._viewcache[0]

    self._oldstr=str
    self._listbox.empty()
    self._listbox.add(self._viewdata)

def _selected(self, event=None):
    self.editor.save()
    self.editor.fill(self._listbox.get())

```

```
class Tester:
```

这是表单外观的定义，格式是 (text, attribute, type)，text 是显示在标签上的文本：
 # attribute 是类的属性，type 是 tkinter 中的类型，或者 doMethod 类型，如果你想要
 # 将一按钮连接到类中的一个方法上。

```

formdef=[('Front:', '_front', StringVar),
          ('Back:', '_backdoor', StringVar),
          ('Descr:', '_description', StringVar),
          ('ID:', '_ID', StringVar),
          ('Status', '_status', IntVar),
          ('Print (demo button)', 'printme', doMethod)]

```

```

def __init__(self, front, back, descr, id, status):
    self._front=front
    self._backdoor=back
    self._description=descr

```



```

        self._ID=id
        self._status=status

    def printme(self):
        print self._front, self._backdoor, self._description,\
              self._ID, self._status

def test():
    values=({'Frontporch', 'backporch', 'my house', 'Uh..', 1),
            ('Roses', 'Violets', 'A garden', 'Aloe vera', 88),
            ('Volvo', 'foo', 'Yack', 'fee', 99),
            ('Python', 'C++', 'lisp', 'C', 52))

    dict={}
    for f,b,d,i,s in values:
        dict[f]=Tester(f,b,d,i,s)

    b=DICTEditor(None, dict, Tester.formdef)
    b.mainloop()
    print "zap"

if __name__=='__main__':
    test()

```

它怎样工作

`dicEdit.py` 为你显示了一个要编辑的字典对象。大量的代码只是用于设置界面，但是如果你需要手动编辑字典，它是值得这么做的。

`test` 函数给了几个数据示例。当从命令行运行 `dicEdit.py` 时它就会弹出来。

如果你运行脚本时出现问题，可以尝试调整它：

自第 211 行：

```

def _init_(self,Master=None,dict=None,fields=None):
    assert (dict is not None)
    assert (fields is not None)

```

自第 229 行：

```

def _init_(self,Master=None,dict=None,formdef=None):

```

```
assert(dict is not None)
assert(formdef is not None)
```

由 Andrew Markebo (Telelogic AB 的软件开发人员) 提供。“实用性在用户的心目中...”

6.4 engine.py

engine.py 实际上有 3 个脚本：两个 Tkinter 脚本和 engine.py 脚本，它们来自第 9 章“服务器”的核心代码。整个源代码包存在于附带的 CD 上。它包括一个自述文件，比许多商用手册更具有综合性。

6.4.1 TkSearch.py

这个脚本把 TkSearch 和 engine.py 连接在一起。

```
#!/usr/bin/env python
...

```

这是一个基于 Tkinter 的统一的 WWW 搜索引擎。

本模块将由 TKSearchUI 模块提供的窗口小部件结构和引擎模块的能力结合起来，创建了一个多线程 Web 搜索引擎 w. a Tkinter 用户接口。

如果你想要实际“浏览”任一搜索的结果，最好已经运行了 Netscape。这个脚本必要时会努力启动 Netscape——参看类 WebBrowser——但是如果已经启动会更快一点。

```
1999 Mitchell S. Chapman
$Id: TkSearch.py,v 1.1 1999/08/22 17:22:34 mchapman Exp mchapman $
...

```

```
__version__ = '$Revision: 1.1 $'
```

```
import string, os, Queue, tempfile
import TkSearchUI
import Tkinter; Tk=Tkinter
import tkMessageBox
import engine

```

```
class WebBrowser:
    def showFile(self, pathname):

```

```

    """在 Web 浏览器中打开路径名字"""
    cmd = "netscape -remote 'openFile(%s)'" % pathname
    status = os.system(cmd)
    if status:
        # Web 浏览器可能不在运行。
        self.launchBrowser()
        os.system(cmd)

def launchBrowser(self):
    """获取作为后台程序运行的 Web 浏览器"""
    pid = os.fork()
    if pid == 0:
        # 这是子进程。
        # 关闭所有父进程可能共享的文件描述符，否则，它将悬挂以等待退出
        for i in range(256):
            try:
                os.close(i)
            except: pass

        # 重定向 stdin、stdout 和 stderr 到 /dev/null 中
        os.open("/dev/null", os.O_RDONLY)
        os.open("/dev/null", os.O_RDWR)
        os.dup2(1, 2)

        # 最后，启动浏览器
        os.execvp("netscape", "netscape")
        raise SystemExit

class SearchManager:
    """这个查询管理器在关闭时让每个线程写入管道。Tkinter 支持文件事件，因此这是一个在编译时更新 Tkinter 的线程安全的方法。"""
    def __init__(self):
        """初始化一新实例"""
        self.queue = Queue.Queue(0)

```

```
self.donePipe = os.pipe()
self.engines = [engine.Excite, engine.InfoSeek, engine.EuroSeek]
# 创建一个文件事件句柄，一旦一线程写到 self.donePipe 即被调用。
Tk.tkinter.createfilehandler(self.donePipe[0],
                              Tk.READABLE, self.collectResultCB)

self.threads = []
self.resultsCB = None

def query(self, queryStr, resultsCB=None):
    """启动一个查询
    如果指定`resultsCB'，当一个新的搜索结果可用时将调用它。"""
    self.resultsCB = resultsCB
    self.threads = []
    for engine in self.engines:
        thread = engine(queryStr, self.threadFinishedCB)
        thread.start()
        self.threads.append(thread)

def cancel(self):
    """取消搜索。"""
    for thread in self.threads:
        if thread.isAlive():
            thread.cancel()

def allDone(self):
    """弄清所有搜索是否完成。"""
    for thread in self.threads:
        if thread.isAlive():
            return 0
    return 1

def threadFinishedCB(self, result):
    """线程结束时调用的回调函数。通过写入自己的 donePipe 将此传入 Tkinter 中。"""
    self.queue.put(result)
    os.write(self.donePipe[1], 'd')
```

```
def collectResultCB(self, *args):
```

```
    """收集线程结果。"""
```

```
    os.read(self.donePipe[0], 1)
```

```
    result = self.queue.get()
```

```
    if self.resultsCB:
```

```
        self.resultsCB(result)
```

```
class T:
```

```
    """这是用于搜索引擎的控制器。"""
```

```
def __init__(self, master=None):
```

```
    """初始化一新实例，如果已经指定，`master`是一将自己的用户接口嵌入其中的窗口小  
    部件"""
```

```
    self.master = master
```

```
    self.searchManager = SearchManager()
```

```
    self.browser = WebBrowser()
```

```
    self.ui = TkSearchUI.T(self.master)
```

```
    self.ui.pack()
```

```
    self.engineState = {}
```

```
    self.engineResult = {}
```

```
    self._connectUIBehaviors()
```

```
def _connectUIBehaviors(self):
```

```
    """Wire up the user interface so it actually does something."""
```

```
    ui = self.ui
```

```
    ui.searchBtn["command"] = self.searchCB
```

```
    ui.stopBtn["command"] = self.stopSearchCB
```

```
    # 现在禁用 stop 按钮。它将在每个搜索启动时启用，在完成所有搜索时不可用
```

```
    ui.stopBtn["state"] = "disabled"
```

```
    ui.quitBtn["command"] = self.quitCB
```

```
    # 为每个可用的搜索引擎添加状态指示器
```

```
self.engineState = {}
for engine in self.searchManager.engines:
    name = engine.__name__
    status = ui.addEngineStatus(name)
    self.engineState[name] = status
    # 为该状态指示器安装 display 按钮
    def resultDisplayCB(self=self, engineName=name):
        self.displayCB(engineName)

    status.displayBtn["command"] = resultDisplayCB
    # 禁用 display 按钮, 它在一些搜索结果确实可用时重新启用
    status.displayBtn["state"] = "disabled"

def searchCB(self, *args):
    """单击 Search 按钮时调用的回叫信号。"""
    queryStr = self.ui.entry.get()
    # 重置所有状态消息。
    for engineState in self.engineState.values():
        engineState.status("Running...")
        # 禁用该引擎的显示状态, 直到它确实显示了结果为止。
        engineState.displayBtn["state"] = "disabled"

    # 禁用 search 按钮, 启用 stop 按钮。
    # 当完成所有搜索时, 重新启用 search 按钮, 而禁用 stop 按钮
    self.ui.searchBtn["state"] = "disabled"
    self.ui.stopBtn["state"] = "normal"

    self.searchManager.query(queryStr, self.resultsCB)

def stopSearchCB(self, *args):
    """单击 stop 按钮时调用的回叫信号。"""
    self.searchManager.cancel()

def quitCB(self, *args):
    """单击 quit 按钮时调用的回叫信号。"""
```

```
if not self.searchManager.allDone():
    self.searchManager.cancel()
self.ui.quitBtn.quit()

def displayCB(self, engineName):
    """单击 Display 按钮时调用的回叫信号。"""
    if self.engineResult.has_key(engineName):
        outname = tempfile.mktemp()
        outf = open(outname, "w", 0)
        outf.write(self.engineResult[engineName].resultStr)
        outf.close()
        self.browser.showFile(outname)
    else:
        print "No status available for %s." % engineName

def resultsCB(self, searchResult):
    """从一个搜索线程接收结果。"""
    engineName = searchResult.engineName
    engineStatus = self.engineState[engineName]
    engineStatus.status(searchResult.finalStatus)
    # 我们具有该引擎的结果, 所以启用它的 display 按钮
    engineStatus.displayBtn["state"] = "normal"

    self.engineResult[engineName] = searchResult

    if self.searchManager.allDone():
        self.ui.stopBtn["state"] = "disabled"
        self.ui.searchBtn["state"] = "normal"

def run(self):
    """启用搜索控制器"""
    self.ui.mainloop()

def main():
    """模块主线(用于独立运行)"""
    t = T()
```

```
t.run()

if __name__ == '__main__':
    main()
```

6.4.2 它怎样工作

TkSearch.py 是这个应用程序的核心 (centerpiece)。它有三个类和一个 main() 函数, 所以这个脚本可以像应用程序一样运行。

整个应用程序的关键部分是 Netscape Navigator。如果浏览器没有打开, 这个脚本就会启动它。运行浏览器的一个问题是如果浏览器没有运行, 那么整个应用程序就会中止。为了避免发生这样的事, WebBrowser.launchBrowser 方法就派生并执行 Netscape 的副本, 而不是使用比较通用的 os.system()。

自从 Python1.5.2 和 Tcl/Tk8.0.5 以来, Tkinter 都是线程安全的。换句话说, 使用后台搜索线程自己进行 UI 更新是可能的。可是, 这个脚本在这些版本出现以前就编写完了, 所以另一个唯一必须克服的挑战是关于线程的 Tkinter 的完全忽略。因为 Tkinter 不提供任何线程的支持, 所以脚本必须支持它, 并且在这里使用的工作区是 Tkinter 的文件事件通知。SearchManager 类创建了一个容纳结果的队列。当每个搜索引擎线程开始活动时, 它都被记录在 SearchManager 的线程列表中。当线程完成时, 结果将移动到结果队列中。

因为不可能对每个操作进行详细解释, 查看 CD 上和 engine.py 包在一起的自述文件。

由 Mitch Chapman 提供!

6.4.3 TkSearchUI

这个脚本是主脚本 engine.py 的第三个组件, 也完全包括 TkSearch.py 和 engine.py。

```
#!/usr/bin/env python
"""
该模块为执行统一的 WWW 搜索提供了用户接口。

1999 Mitchell S. Chapman
$Id: TkSearchUI.py,v 1.1 1999/08/22 17:22:50 mchapman Exp mchapman $
"""

__version__ = "$Revision: 1.1 $"

import Tkinter; Tk=Tkinter
```



```

class EngineStatus:
    """该类给用户接口提供一搜索引擎状态指示器。"""

    def __init__(self, master=None, engineName="Engine"):
        """初始化一新实例。"""
        self.master = master
        self.engineName = engineName
        f = self.frame = Tk.Frame(self.master, relief="flat", bd=0)
        text = "%s:" % self.engineName
        l = self.label = Tk.Label(f, text=text, width=len(text),
                                   anchor="e")
        s = self.statusLabel = Tk.Label(f, width=12, anchor="w")
        btn = self.displayBtn = Tk.Button(f, text="Display...")
        l.pack(side="left", fill="none", expand="no")
        s.pack(side="left", fill="x", expand="yes")
        btn.pack(side="left", fill="none", expand="no")

    def pack(self, **kw):
        """将自身封装到主控中。`kw`提供可选的封装参数"""
        params = {'side': 'top', 'fill': 'x', 'expand': 'no'}
        params.update(kw)
        apply(self.frame.pack, (), params)

    def status(self, msg=""):
        """更新自己的状态信息"""
        self.statusLabel["text"] = msg
        self.statusLabel.update_idletasks()

class T:
    """该类定义了一个用于 WWW 搜索的用户接口层。它几乎没有定义任何行为。"""

    def __init__(self, master=None, title="Search The Web"):
        """初始化一个新实例。
        如果指定`master`，是将该 UI 嵌入其中的一个窗口小部件"""
        self.master = master

```

```

self.title = title

f = self.frame = Tk.Frame(master, relief="flat", bd=0)
# 将标题置于自身的高层窗口上。
f.winfo_toplevel().wm_title(self.title)

self._addControlsFrame()
self.engineStates = []
self._addStatusFrame()
self._addButtonBox()

def _addControlsFrame(self):
    """添加一个包含搜索控制的框架。"""
    cf = self.controlsFrame = Tk.Frame(self.frame, relief="raised", bd=1)

    l = Tk.Label(cf, text="Query:", anchor="e")
    l.pack(side="left", fill="none", expand="no")

    e = self.entry = Tk.Entry(cf, width=32)
    e.pack(side="left", fill="x", expand="yes")

    b = self.searchBtn = Tk.Button(cf, text="Search")
    b.pack(side="left", fill="none", expand="no")

    b = self.stopBtn = Tk.Button(cf, text="Stop")
    b.pack(side="left", fill="none", expand="no")

    cf.pack(fill="both", expand="yes")

def _addStatusFrame(self):
    """添加一个在其中显示搜索状态的框架。"""
    rf = self.resultsFrame = Tk.Frame(self.frame,
                                      relief="raised", bd=1)
    rf.pack(fill="both", expand="yes")

```

```
def addEngineStatus(self, engineName):
    """客户调用此函数以添加一个搜索引擎状态指示器。"""
    result = EngineStatus(self.resultsFrame, engineName)
    self.engineStates.append(result)
    # 进行调整, 以便所有的状态指示器都具有相同的宽度。
    wMax = 0
    for status in self.engineStates:
        wMax = max(wMax, status.label["width"])
    for status in self.engineStates:
        status.label["width"] = wMax

    result.pack()
    return result

def _addButtonBox(self):
    """在 UI 的主窗口底部添加一个按钮框。"""
    bbf = self.buttonBoxFrame = Tk.Frame(self.frame, relief="raised",
        bd=1)

    qb = self.quitBtn = Tk.Button(bbf, text="Quit")
    qb.pack(side="left", expand="yes")

    bbf.pack(side="bottom", fill="x", expand="no")

def pack(self, **kw):
    """将自身的 UI 封装到它的主控中。如果指定`kw`, 则包含说明如何封装的关键字参数。"""
    packing = {'fill': 'both', 'expand': 'yes'}
    packing.update(kw)
    apply(self.frame.pack, (), packing)

def mainloop(self):
    """运行主事件循环。"""
    self.frame.mainloop()
```

```
def main():
    """模块主线(用于独立运行)"""
    t = T()
    stati = {}
    for engine in ["EuroSeek", "InfoSeek", "Excite"]:
        status = t.addEngineStatus(engine)
        stati[engine] = status
    t.pack()
    t.mainloop()

if __name__ == "__main__":
    main()
```

6.4.4 它怎样工作

TkSearchUI.py 是整个 engine.py 应用程序的 GUI 核心。虽然如此,它只有不多的操作代码。

从用户的观点来看,类 T 在整个应用程序中从事最重要的工作。T 是生成输入字段、Search 按钮、Quit 按钮和 Stop 按钮的类。

main(), 尽管它的稍微极度重要的名字,只创建 T 的一个实例,添加几个 EngineStatus 显示实例,然后显示作为结果的窗口小部件层次。事实上,main()唯一的目的是让模块维护人员检查窗口小部件层次布局。

要得到更多的关于在这个脚本中不同的语句是如何工作的信息,查看第 4 章关于 Tkinter 窗口小部件的讨论。要得到更多关于 engine.py 的信息,查看附带的 CD 上 engine.py 包中的自述文件。

由 Mitch Chapman 提供。

第 7 章 数据库

本章要点:

- nickname.py
- dbase3.py

本章有介绍 Python 的数据库函数的脚本。Python 有几个能存储 Python 对象的标准模块: pickle、cpickle、marshal 和 shelve 只是少数几个。这些能做为简单数据库使用。为了得到更高级的数据库, 你需要一些不同的模块。本章中的一个模块, 例如, 访问 dBASE 文件。但为了得到更强大的功能, Python 可以在 SQL 中直接使用更复杂的模块。这些模块中的大部分可以免费从 www.python.org 下载得到。

本章只是对基本操作的介绍。如果你需要使用 Python 访问 Oracle、ODBC、Informix 或类似系统, 你将会在 www.python.org 找到许多更加高级的材料。特别是阅读 Joel Shprentz 的论文“Persistent Storage of Python Objects in Relational Databases”。也可以在 www.python.org 上得到。正如 Shprentz 指出的那样, 因为行和列的关系域 (relational universe) 并不是完美地适合 Python 的对象, 所以 Python 不是生来就对数据库非常友好。但是有一种方法可以使用 Python 访问关系数据库: 永久性存储模块。在 Python 站点可以得到它的详细资料。

7.1 nickname.py

这是一个非常简单的脚本, 像类字典对象一样存储绰号, 它使用 shelve 模块。

```
#!/usr/bin/python

import shelve

def makeshelf(name):
    name_of_file=shelve.open(name)
    while 1:
        shelfkey = raw_input("What's your name? Type Q to quit. ")
        if shelfkey != "Q":
            value = raw_input("What's your nickname?: ")
            name_of_file[shelfkey]=value
```

```

else:
    print "Thanks. Got the data and it's properly stored. Here's what you
entered: "

    keylist=name_of_file.keys()
    for keyword in keylist:
        printout=name_of_file[keyword]
        print "Your name is ",keyword, "and your nickname is", printout
    name_of_file.close()
    return

if __name__=="__main__":
    name=raw_input("File name please: ")
    makeshelf(name)

```

它怎样工作

除非你为保存脚本外部的数据做好了准备，否则当脚本结束时，数据就会消失。

在 `nickname.py` 中，首先会请求一个文件名。这个文件名传递到 `makeshelf()` 函数，如果文件还不存在就创建文件，或者如果文件已经存在就打开该文件。

然后，这个脚本进入 `while` 循环，通过键入大写字母 `Q` 能断开循环。名字和绰号连续存储在 `shelf` 对象中。`shelf` 对象很方便，因为它像字典一样。相同的语句可以为这两种情况工作，所以如果你比较熟悉字典，则会做得很好。这些行如下：

```

shelfkey = raw_input("What's your name? Type Q to quit. ")
if shelfkey != "Q":
    value = raw_input("What's your nickname?: ")
    name_of_file[shelfkey]=value

```

创建两个变量，一个适用于名字，另一个适用于绰号。然后这些变量分别成为关键字和每个字典条目的数据值。当编写这两个变量时，循环重新开始。当用户结束时，`Q` 停止循环并使脚本打印出结果。如果你正在输入数千个项目，这种途径不实用，所以如果正在做大量操作可以考虑把它放在一边。

7.2 dbase3.py

这个脚本定位于在简单持久性模块和完全数据库访问模块之间的某些地方。

...

为读取 `dBase3` 文件格式定义一个类。

格式定义如下:

dBASE III 数据库文件结构(参看最后的属性)

dBASE III 数据库文件结构包括一个头和数据记录。布局如下:

dBASE III 数据库文件头:

字节	容量	意思
0	1 字节	dBASE III 版本号 (03H: 非 .DBT 文件) (83H: .DBT 文件)
1-3	3 字节	最后更新的日期 (YY MM DD) 以二进制格式
4-7	32 位数	数据文件中的记录号
8-9	16 位数	头结构的长度
10-11	16 位数	记录长度
12-31	20 字节	保留字节 (1.00 版)
32-n	每个 32 字节	字段描述符数组 (参看下面)
n+1	1 字节	0DH 作为字段终止符

字段描述符:

字节	容量	意思
0-10	11 字节	ASCII 零填充字段名
11	1 字节	ASCII 码字段类型 (C N L D 或 M)
12-15	32 位数	字段数据地址 (地址按内存设置)
16	1 字节	二进制字段长度
17	1 字节	二进制格式字段
18-31	14 字节	保留字节 (1.00 版)

数据记录布局如下:

1. 数据记录用一个字节作前导, 如果记录未被删除, 这个字节是空格 (20H), 如果删除, 是星号 (2AH)。
2. 数据字段封装到了没有字段分隔符或记录终止符的记录中。
3. 数据类型以 ASCII 格式保存如下:

数据类型	数据记录保存
字符	(ASCII 字符)
数值	- . 0 1 2 3 4 5 6 7 8 9
逻辑	? Y y N n T t F f (? 在没有初始化时)
信息	(这 10 个数字代表 .DBT 块号)
日期	(YYYYMMDD 格式中的 8 个数字, 如 19840704 代表 July 4, 1984)

上述信息直接来自于 Ashton-Tate 论坛, 也可以在 Ashton-Tate 的高级程序员指南中找到。

在分别由 dBASE III 和 dBASE III Plus 创建的文件间会稍有差异。早期的文件中，在头指示器结束 \$0D 和数据的开始前有一个 ASCII 码 NUL 空白。这个 NUL 在 Plus 版中没有，使得 Plus 版的头部比同样结构的 III 文件少一个字节。

取自 DBF.PAS 1.3 版的 Pascal 程序

Copyright (C) 1986 By James Troutman

CompuServe PPN 74746,1567

授权允许将这些程序用于非商业目的。

...

```
def Build32BitInteger(four_byte_string):
    '''这将一字符串解压为一 32 位整数。'''
    ...

    if len(four_byte_string) != 4:
        raise "Bad data", "String not 4 bytes long"

    s0 = ord(four_byte_string[0])
    s1 = ord(four_byte_string[1])
    s2 = ord(four_byte_string[2])
    s3 = ord(four_byte_string[3])

    return (s3 << 24) | (s2 << 16) | (s1 << 8) | s0
```

```
def StripNulls(str):
    '''在第一个后缀空处截断字符串，返回截断字符串。'''
    ...

    if len(str) < 1: return str
    first_null = -1
    for ix in xrange(len(str)):
        if ord(str[ix]) == 0:
            first_null = ix
            break
    if first_null == -1:
        return str
    else:
        return str[0:first_null]
```

```

class dBase3:
    '''读取 dBASE3 文件的类。注意只读头部信息，并且有足够的功能遍历字段并解释它们。如果
    想要向 dBASE 文件回写，必须写一个子类。'''

    def __init__(self, file_name):
        '''我们输入了文件名，所以打开了它。接下来要读取头部的前 20 字节，获取基本数据（参
        看上面的 dBASE3 描述）。一旦知道了头部的大小就可全部获取它。然后读入并解析字段信息。'''

        import string
        self.file_name = file_name
        try:
            self.fp = open(file_name, "rb")
        except:
            raise "Bad data", "Couldn't open \"%s\" " % file_name
        self.header = self.fp.read(20) # 读取头
        if not self.header:
            raise "Bad data", "File is empty"
        if len(self.header) != 20:
            raise "Bad data", "Missing first 20 bytes in file header"
        self.version_number = ord(self.header[0])
        self.last_update_yy = ord(self.header[1])
        self.last_update_mm = ord(self.header[2])
        self.last_update_dd = ord(self.header[3])
        self.num_records = Build32BitInteger(self.header[4:8])
        self.header_length = (ord(self.header[9]) << 8) | \
                               ord(self.header[8])
        self.record_length = (ord(self.header[11]) << 8) | \
                               ord(self.header[10])
        # 现在返回并读取包括字段说明的整个头部
        self.fp.seek(0)
        self.header = self.fp.read(self.header_length)
        self.reserved_bytes = self.header[12:32]
        # 估计我们必须有多少个字段描述符
        self.num_fields = (self.header_length - 33)/32
        # 现在读取字段信息
        self.fields = {}

```

```

for ix in range(self.num_fields):
    start = 32 + ix*32
    stop = start + 32 + 1
    field_data = self.header[start : stop]
    field = []
    # 获得字段名
    str = string.strip(StripNulls(field_data[0:11]))
    field.append(str)
    # 获得字段类型字符
    field.append(field_data[11])
    # 获得字段数据地址
    field.append(StripNulls(field_data[12:16]))
    # 获得字段长度
    field.append(ord(field_data[16]))
    # 获得字段的十进制数
    field.append(ord(field_data[17]))
    # 获得字段保留的字节
    field.append(StripNulls(field_data[18:32]))
    self.fields.append(field)

def DumpHeader(self):
    spaces = 30
    template = "%%-s %d" % spaces
    template1 = "%%-s 0x%02x" % spaces
    print "File: %s" % self.file_name
    print template1 % ("Version info", self.version_number),
    if self.version_number == 0x83:
        print " (with a .DBT memo file)"
    else:
        print " (without a .DBT memo file)"
    print template % ("Last update year", self.last_update_yy+1900)
    print template % ("Last update month", self.last_update_mm)
    print template % ("Last update day", self.last_update_dd)
    print template % ("Number of records", self.num_records)
    print template % ("Header length", self.header_length)

```

```

print template % ("Record length", self.record_length)
print template % ("Number of fields", self.num_fields)
print "\nDump of field info:\n"
print "    Num    Name          Type    Length"
print "    ---    -"
for ix in range(self.num_fields):
    print "    %2d" % ix,          # 字段号
    s = self.fields[ix]
    print "    %-12s" % s[0],      # 名称
    print "    %-1s" % s[1],      # 类型
    #print "    \"%-4s\" " % s[2],
    print "    %3d" % s[3]        # 长度
    #print "    %3d" % s[4],
    #print "    %-14s" % s[5]

def GetFileHandle(self):
    return self.fp

def _get_record(self, record_num):
    '''将以字符串返回指定的记录。记住第一个字符是 '.' 或 '*', 表明记录是否删除。'''
    if record_num < 0 or record_num > (self.num_records - 1):
        raise "Bad data", "Record number out of bounds"
    self.fp.seek(self.header_length + record_num*self.record_length)
    str = self.fp.read(self.record_length)
    return str

def GetRecordAsList(self, record_num):
    '''将以字符串列表返回指示的记录。列表中的位置和字段号相同。'''
    import string
    record = []
    str = self._get_record(record_num)
    if len(str) != self.record_length:
        raise "Bad data", "Record len is %d, not %d" % \
            (len(str), self.record_length)
    offset = 1 # 1 让我们越过指示删除的第一个字节

```

```

for ix in xrange(self.num_fields):
    name = self.fields[ix][0]
    type = self.fields[ix][1]
    length = self.fields[ix][3]
    field = str[offset : offset + length]
    record.append(field)
    offset = offset + length # 在下一个字段返回记录开始时的位置区域

```

```

def GetRecordAsDictionary(self, record_num):

```

'''将以字典返回指示的记录。关键字是字段名，字段数据将被转换成恰当的类型：数字为整数或双精度型，日期为 YYYYMMDD 字符串，逻辑值和字符串仍为字符串。'''

```

import string
record = {}
str = self._get_record(record_num)
if len(str) != self.record_length:
    raise "Bad data", "Record len is %d, not %d" % \
        (len(str), self.record_length)
offset = 1 # 1 让我们越过指示删除的第一个字节
for ix in xrange(self.num_fields):
    name = self.fields[ix][0]
    type = self.fields[ix][1]
    length = self.fields[ix][3]
    field = str[offset : offset + length]
    if type == 'C' or type == 'L' or type == 'D':
        record[name] = field
    elif type == 'N':
        # 如果在字段中有 '.', 那么要解释它为双精度型，否则会将其解释为一个有符号
        # 的整数
        value = 0
        field = string.strip(field)
        if field != "":
            if '.' in field:
                value = string.atof(field)
            else:
                value = string.atoi(field)

```

```

        record[name] = value
    elif type == 'M':
        # 我们忽略所有 memo 字段
        pass
    else:
        raise "Bad data", "Unrecognized field type"
    offset = offset + length # 在下一个字段返回记录开始时的位置区域

def DumpAsDelimited(self, no_header = 0, delimit_char = "\t", ws_strip =
1):
    '''用由 delimit_cher 中的字符串划界的每个字段将数据库文件的每个记录打印到
    stdout。如果 ws_strip 是真, 将删除任何前导或尾随空格。如果 no_header 是假, 将打印出字段名'''
    import sys, string
    if not no_header:
        # 打印头
        for ix in xrange(self.num_fields):
            field_name = self.fields[ix][0]
            sys.stdout.write(field_name)
            if ix != self.num_fields - 1:
                sys.stdout.write(delimit_char)
            sys.stdout.write("\n")
        # 打印每个记录
        for recnum in xrange(self.num_records):
            record = self.GetRecordAsList(recnum)
            for ix in xrange(self.num_fields):
                field = record[ix]
                if ws_strip:
                    field = string.strip(field)
                sys.stdout.write(field)
                if ix != self.num_records - 1:
                    sys.stdout.write(delimit_char)
            sys.stdout.write("\n")

if __name__ == '__main__':

```

```
import sys, string
if len(sys.argv) < 2:
    print "Usage: dbase3 <dbase_file>"
    print " Will dump the header and data of a dbase3 file."
    sys.exit(1)
db = dBase3(sys.argv[1])
db.DumpHeader()
db.DumpAsDelimited()
```

它怎样工作

好家伙，这是一个不缺乏注解行的脚本！

脚本的大部分是一个类，**dBase3**。当它被实例化时，**dBase3** 试图打开指定的文件。打开语句的文件在 **try** 序列内部，如果文件不存在，就会让脚本进行出错捕捉。如果你试图打开文件而不确信文件是否在那时，在 **try** 序列内部放置文件打开程序是一个极好的策略。当然，有时你知道文件就在那，因为你已经把它放在那了。**CGI** 访问数据库文件就是一个例子。但是尽管那样，你还能用 **try** 序列打开文件，只是预防脚本会发生某些错误。

使用这个脚本，你可以用多种方法得到 **dBaseIII** 记录。可以得到一个头文件的转储映像、不限格式的记录转储映像、记录的字符串表述、记录的列表表述或记录的字典表述。

正如注释中所说的那样，这个脚本不会对 **dBaseIII** 文件进行写操作，它只会读取它们。由匿名者提供。

第 8 章 数学/科学函数

本章要点：

- donutil.py
- julian.py
- roman.py
- stats.py

本章你将会找到一些执行基本数学和科学函数的极好却不多的脚本。有几个脚本阐明了怎样使用数学模块，它是标准库中最方便的模块之一。

这些脚本简单而且短。自由描绘 Python 时它们是很好的，但是陈列出 Python 完全的性能恐怕要花不少的光年。由于如些多的数学和科学模块可以使用，Python 将成为科学上的（至少是部分）程序设计员的更加通用的语言。

例如，这里有一些模块可以下载得到，就像 www.python.org 上记录的那样：

- PyEphem。计算天文体的位置。
- PYML。数学接口。
- NumPy。提供高性能多维数值数组。

8.1 donutil.py

在这个你发现有用的脚本中有很少的几个函数。它显示标尺，计算风冷却、用图解表示弧度和角度并且做一些系统工作。它不是简单的分类，却如何是好？

...

用 python 写的杂项程序：

Ruler	返回一个标尺
TensRuler	伴随 Ruler() 的 10 的标尺
WindChillInDegF	计算 OAT 给的风冷却和风速
Deg2Rad	将度转换为弧度
Rad2Deg	将弧度转换为度
IsFile	如果参数是文件返回真
IsDirectory	如果参数是目录返回真

SpellCheck

检查字典中的单词列表

...

```
def Ruler(size = 79, type = 1):
```

```
    ...
```

返回到标尺字符串中的函数

第一类标尺:

```

          1           2           3           4           5           6           7
1234567890123456789012345678901234567890123456789012345678901234567890
```

第二类标尺

```

          1           2           3           4           5           6           7
-----+-----|-----+-----|-----+-----|-----+-----|-----+-----|
```

第三类标尺:

```

-----+-----|-----+-----|-----+-----|-----+-----|-----+-----|
```

第四类标尺:

```

-----+-----|-----+-----|-----+-----|-----+-----|-----+-----|
```

```
    ...
```

```
if size < 1:
```

```
    raise "Bad data", "Ruler: size < 1"
```

```
str = ""
```

```
if type == 1:
```

```
    str = TensRuler(size)
```

```
    base = "1234567890"
```

```
elif type == 2:
```

```
    str = TensRuler(size)
```

```
    base = "-----+-----|"
```

```
    pass
```

```
elif type == 3:
```

```
    base = "-----+-----|"
```

```

elif type == 4:
    base = "-----|"
else:
    raise "Bad data", "Ruler: type not between 1 and 3"
num_repeats = (size/10 + 2) * 10
tmpstr = base * num_repeats
str = str + tmpstr[:size]
return str

```

```

def TensRuler(size):
    if size >= 10:
        str = ""
        for ix in range(1, 1 + size/10):
            str = str + "%10d" % (ix * 10)
        str = str + "\n"
    return str

```

```

def WindChillInDegF(wind_speed_in_mph, air_temp_deg_F):
    '''暴露的人皮肤的风冷却，表达为风速（英里/小时）和温度（华氏温度）的函数。取自代码
    片断集。'''

```

```

    import math
    if wind_speed_in_mph < 4:
        return air_temp_deg_F * 1.0
    return (((10.45 + (6.686112 * math.sqrt(1.0*wind_speed_in_mph))) \
        - (.447041 * wind_speed_in_mph)) / 22.034 * \
        (air_temp_deg_F - 91.4)) + 91.4)

```

```

def Deg2Rad(degrees):
    import math
    return degrees/180.0 * math.pi

```

```

def Rad2Deg(radians):
    import math
    return radians/math.pi * 180.0

```

```
def SpellCheck(input_list, word_dictionary, case_is_not_important = 1):
```

```
    ...
```

本模块提供 `SpellCheck()` 函数，它将输入作为在 `input_list` 和字典 `word_dictionary` 中进行拼写检查的单词列表（它是一个字典而非一个允许快速访问的列表，字典值可以是空字符串——重要的是在有关键字存在）。它返回不在 `word_dictionary` 中的 `input_list` 中的任何单词。

```
    ...
```

```
import string
misspelled = []
if len(input_list) == 0:
    return []
if len(word_dictionary) == 0:
    raise 'SpellCheck: word_dictionary parameter is empty'
for ix in xrange(len(input_list)):
    if case_is_not_important:
        word = string.lower(input_list[ix])
    if not word_dictionary.has_key(word):
        misspelled.append(word)
return misspelled
```

```
def Keep(str, keep_chars):
```

```
    "只将指定的字符保存在字符串中"
```

```
    strlength = len(str)
    if strlength == 0 or len(keep_chars) == 0:
        return ""
    count = 0
    outstring = ""
    while count < strlength:
        if str[count] in keep_chars:
            outstring = outstring + str[count]
            count = count + 1
    return outstring
```

```
def Remove(str, remove_chars):
```

```
    "从字符串中删除指定的字符。"
```

```
    strlength = len(str)
```

```

if strlength == 0:
    return ""
if len(remove_chars) == 0:
    return str
count = 0
outstring = ""
while count < strlength:
    if str[count] not in remove_chars:
        outstring = outstring + str[count]
    count = count + 1
return outstring

```

它怎样工作

这里最有趣的函数是用风冷却计算器和弧度转换。它们都使用数学模块。

注意每个函数都要分别输入数学模块。如果数学模块在任何函数的外部输入就不是必需的。你能会想是否输入了，脚本中必须输入的函数内的数学模块只有 `Deg2Rad`，那就错了！解释程序扫描整个模块并输入它所需要的，即使一个输入的模块为了所有的函数再输入一次。

通过使用圆括号，能把多个函数压缩到一行上进行嵌套操作。优先顺序是从内部到外部：

```
return (((10.45 + (6.686112 * math.sqrt(1.0*wind_speed_in_mph)))\
```

数学模块有几个普通的数学函数，例如正切 (`tangent`)、平方根 (`square root`) 和阶乘 (`exponent`)。另一个模块，`cmath`，用于复数。数学模块也有两个常量：`pi` 和 `e`。

匿名者提供

这里提供了让你亲自演示和判断的例子：Python 让你在方法和函数中重新声明一个变量为全局变量，如下所示：

```

x=1
def a_list_is_born():
    global x
    x=['a' , 'list' ]

```

这样允许全局变量 `x` 在函数中保持为全局变量，并且这是合法的。然而，你想要冒险改变函数或方法中的全局变量吗？有些 Python 最资深的人劝告不要这样做，但是许多 Python 程序员还是这样做。

8.2 julian.py

这个脚本为你指定的任何日期都提供了儒略 (Julian) 日。脚本能使它们作为整数或真

实的天文时期返回。它也能把 Julian 日转换为 Gregorian 日期并做几件有趣的事。如果你不了解 Julian 日，向下移动查看“它怎样工作”的解释。

这个脚本准备作为一个模块使用。作为主群组运行，它启动一个测试序列。也要注意它不能从系统中获取当前日期，你必须把日期作为参数来传递。

'''

由 Meeus 写的 Julian 日程序“计算机天文计算公式”。

程序为：

Julian(month, day, year)	整数 Julian 日号
JulianAstro(month, day, year)	天文 Julian 日号
JulianToMonthDayYear(julian_day)	返回 month, day, year 元组
DayOfWeek(month, day, year)	0 = Sunday
DayOfYear(month, day, year)	1~365 (在闰年中为 366)
IsValidDate(month, day, year)	如果日期是有效的 Gregorian 返回真
IsLeapYear(year)	如果 year 是闰年，返回真
NumDaysInMonth(month, year)	

JulainAstro 函数返回天文格式，这是以浮点数返回的。天文 Julain 日在格林威治中午开始。Julain() 函数返回较普通的 Julain 日为一整数；它通过在天文格式的基础上加 0.55 取整数得到。

警告：通常函数 do_not_检查它们引入的参数。

Don Peterson 1998 年 5 月 30 日

'''

```
def NumDaysInMonth(month, year):
    if month == 2:
        if IsLeapYear(year):
            return 29
        else:
            return 28
    elif month == 9 or month == 4 or month == 6 or month == 11:
        return 30
    elif month == 1 or month == 3 or month == 5 or month == 7 or \
        month == 8 or month == 10 or month == 12:
```

```

        return 31
    else:
        raise "Bad month"

def JulianToMonthDayYear(julian_day):
    if julian_day < 0: raise "Bad input value"
    jd = julian_day + 0.5
    Z = int(jd)
    F = jd - Z
    A = Z
    if Z >= 2299161:
        alpha = int((Z - 1867216.26)/36254.25)
        A = Z + 1 + alpha - int(alpha/4)
    B = A + 1524
    C = int((B - 122.1)/365.25)
    D = int(365.25 * C)
    E = int((B - D)/30.6001)
    day = B - D - int(30.6001 * E) + F
    if E < 13.5:
        month = int(E - 1)
    else:
        month = int(E - 13)
    if month > 2.5:
        year = int(C - 4716)
    else:
        year = int(C - 4715)
    return month, day, year

def DayOfYear(month, day, year):
    if IsLeapYear(year):
        n = int((275*month)/9 - ((month + 9)/12) + int(day) - 30)
    else:
        n = int((275*month)/9 - 2*((month + 9)/12) + int(day) - 30)
    if n < 1 or n > 366: raise "Internal error"
    return n

```

```
def DayOfWeek(month, day, year):
    julian = int(JulianAstro(month, int(day), year) + 1.5)
    return julian % 7

def IsLeapYear(year):
    if (year % 400 == 0) or (year % 4 == 0 and year % 100 != 0):
        return 1
    else:
        return 0

def IsValidDate(month, day, year):
    '''如果 year 晚于 1752, 并且 month 和 day 有效, 返回真。
    ...

    if month < 1 or month > 12: return 0
    if int(month) != month : return 0
    if year < 1753 : return 0
    if day < 1.0 : return 0
    if int(day) != day:
        if month == 2:
            if IsLeapYear(year):
                if day >= 30.0: return 0
            else:
                if day >= 29.0: return 0
        elif month == 9 or month == 4 or month == 6 or month == 11:
            if day >= 31.0 : return 0
        else:
            if day >= 32.0 : return 0
    else:
        if month == 2:
            if IsLeapYear(year):
                if day >= 29 : return 0
            else:
                if day >= 28 : return 0
        elif month == 9 or month == 4 or month == 6 or month == 11:
```

```

        if day >= 30      : return 0
    else:
        if day >= 31      : return 0
    return 1

```

```
def JulianAstro(month, day, year):
```

“注意 day 既可以是整数，也可以是浮点数”

```

    if month < 3:
        year = year - 1
        month = month + 12
    julian = int(365.25*year) + int(30.6001*(month+1)) + day + 1720994.5
    tmp = year + month / 100.0 + day / 10000.0
    if tmp >= 1582.1015:
        A = year / 100
        B = 2 - A + A/4
        julian = julian + B
    return julian * 1.0

```

```
def Julian(month, day, year):
```

```
    return int(JulianAstro(month, day, year) + 0.55)
```

```
def Test():
```

```

    if Julian(12, 31, 1989)      != 2447892    : raise "TestError"
    if Julian(1, 1, 1990)        != 2447893    : raise "TestError"
    if Julian(7, 4, 1776)        != 2369916    : raise "TestError"
    if Julian(2, 29, 2000)       != 2451604    : raise "TestError"
    if JulianAstro(1, 27.5, 333)  != 1842713.0 : raise "TestError"
    if JulianAstro(10, 4.81, 1957) != 2436116.31: raise "TestError"
    if DayOfWeek(11, 13, 1949)    != 0         : raise "TestError"
    if DayOfWeek( 5, 30, 1998)    != 6         : raise "TestError"
    if DayOfWeek( 6, 30, 1954)    != 3         : raise "TestError"
    if DayOfYear(11, 14, 1978)    != 318       : raise "TestError"
    if DayOfYear( 4, 22, 1980)    != 113       : raise "TestError"

```

```
month, day, year = JulianToMonthDayYear(2436116.31)
```



```
if month != 10 : raise "TestError"
if year != 1957 : raise "TestError"
if abs(day - 4.81) > .00001 : raise "TestError"
month, day, year = JulianToMonthDayYear(1842713.0)
if month != 1 : raise "TestError"
if year != 333 : raise "TestError"
if abs(day - 27.5) > .00001 : raise "TestError"
month, day, year = JulianToMonthDayYear(1507900.13)
if month != 5 : raise "TestError"
if year != -584 : raise "TestError"
if abs(day - 28.63) > .00001 : raise "TestError"

if NumDaysInMonth( 1, 1999) != 31 : raise "TestError"
if NumDaysInMonth( 2, 1999) != 28 : raise "TestError"
if NumDaysInMonth( 3, 1999) != 31 : raise "TestError"
if NumDaysInMonth( 4, 1999) != 30 : raise "TestError"
if NumDaysInMonth( 5, 1999) != 31 : raise "TestError"
if NumDaysInMonth( 6, 1999) != 30 : raise "TestError"
if NumDaysInMonth( 7, 1999) != 31 : raise "TestError"
if NumDaysInMonth( 8, 1999) != 31 : raise "TestError"
if NumDaysInMonth( 9, 1999) != 30 : raise "TestError"
if NumDaysInMonth(10, 1999) != 31 : raise "TestError"
if NumDaysInMonth(11, 1999) != 30 : raise "TestError"
if NumDaysInMonth(12, 1999) != 31 : raise "TestError"

if NumDaysInMonth( 1, 2000) != 31 : raise "TestError"
if NumDaysInMonth( 2, 2000) != 29 : raise "TestError"
if NumDaysInMonth( 3, 2000) != 31 : raise "TestError"
if NumDaysInMonth( 4, 2000) != 30 : raise "TestError"
if NumDaysInMonth( 5, 2000) != 31 : raise "TestError"
if NumDaysInMonth( 6, 2000) != 30 : raise "TestError"
if NumDaysInMonth( 7, 2000) != 31 : raise "TestError"
if NumDaysInMonth( 8, 2000) != 31 : raise "TestError"
if NumDaysInMonth( 9, 2000) != 30 : raise "TestError"
if NumDaysInMonth(10, 2000) != 31 : raise "TestError"
```

```
if NumDaysInMonth(11, 2000) != 30 : raise "TestError"
if NumDaysInMonth(12, 2000) != 31 : raise "TestError"

if __name__ == "__main__":
    Test()
```

它怎样工作

1540 年, Joseph Justus Scalinger 计算出一段时间, 称为“Julian 周期”。它实际上是从 3 个小循环衍生而来的大循环。Julian 周期有 7980 年长。它的名字是混乱的, 因为它和 Julian 日历有点或毫无关系。

Julian 周期的整个特点是有一个用于参考远古时间的单一的固定系统, 而不是依赖许多不同纪元和朝代的 wobbly 日历的拼凑物。使用 Julian 循环, Scalinger 能通过使用符合 3 个内部循环的周期的 3 个数字的组合来引用任何过去的日期。听起来错综复杂, 而且它就是这样的, 但是比试图协调旧的日历要好得多。

天文学家 John Herschel 编写了一个比较好的 Julian 循环。1849 年, 他公布了 Julian 循环想法的延伸, 主张“Julian 日”。似乎是天文学家为了推算事件需要一种方法把过去几百年的观察和今天的观察联系起来。精确某事发生的时间不是问题。较多的问题是相同事件之间流逝了多少时间。但是没有单个日历, 你怎样比较 Chaldean 时期的观察和上星期二发生的事呢? 你需要一种方法一贯地比较时期, 而且在 Julian 日方便进来的地方。你所需要知道的是事情发生的 Julian 日; 然后计算出下一个事件发生的 Julian 日并减去上一个 Julian 日。计数系统的零基线是 January 1, 4713 B.C.E. 当我写到这时, 今天的 Julian 日是 2451401。如果就在这时进行转换, 你会得到一个半日的奇怪结果, 这是由 Julian 天文日开始和结束的时间决定的。

即使 Python 变量不必声明, 你考虑初始化它们来避免 NameError 也是很好的。这样做可以消除异义。例如, 在脚本的顶端有:

```
x=y./2
```

由于 y 没有定义, 所以会产生一个 y 的 NameError。这可以在启动主代码之前, 通过初始化 x 和 y 来避免, 几乎可以用任何方法来初始化, 例如, x=y=10。如果这一行在函数定义中, 你可以用下面的方法很容易地初始化变量:

```
def factor(x=10, y=10):
```

使用这个脚本, 你能得到一个整数 Julian 日或一个天文 Julian 日。你也可以走另一条路并从 Julian 日中获得 Gregorian 日期。这个模块不使用任何其他输入的模块。每件事都可以用基本的 Python 语句和操作来处理。

匿名者提供

8.3 roman.py

曾经想要一种把老电影上的罗马数字版权注意事项，翻译成有点像 20 世纪的东西的快速方法。在我们最早的算术课上我们已经学会如何转换罗马数字，但是我们中有多少人记住了呢？

```
import re
```

```
iv = re.compile("IV")
```

```
ix = re.compile("IX")
```

```
xl = re.compile("XL")
```

```
xc = re.compile("XC")
```

```
cd = re.compile("CD")
```

```
cm = re.compile("CM")
```

```
i = re.compile("I" )
```

```
v = re.compile("V" )
```

```
x = re.compile("X" )
```

```
l = re.compile("L" )
```

```
c = re.compile("C" )
```

```
d = re.compile("D" )
```

```
m = re.compile("M" )
```

```
def RomanNumeralsToDecimal(roman_string):
```

```
    import re, string
```

```
    str = string.upper(roman_string)
```

```
    exp = ""
```

```
    if iv.search(str) != None: str = iv.sub("+4", str)
```

```
    if ix.search(str) != None: str = ix.sub("+9", str)
```

```
    if xl.search(str) != None: str = xl.sub("+40", str)
```

```
    if xc.search(str) != None: str = xc.sub("+90", str)
```

```
    if cd.search(str) != None: str = cd.sub("+400", str)
```

```
    if cm.search(str) != None: str = cm.sub("+900", str)
```

```
    if i.search(str) != None: str = i.sub("+1", str)
```

```
    if v.search(str) != None: str = v.sub("+5", str)
```

```
    if x.search(str) != None: str = x.sub("+10", str)
```

```
    if l.search(str) != None: str = l.sub("+50", str)
```

```
if c.search(str) != None: str = c.sub("+100", str)
if d.search(str) != None: str = d.sub("+500", str)
if m.search(str) != None: str = m.sub("+1000", str)
exec "num = " + str
return num
```

```
def DecimalToRomanNumerals(base10_integer):
    '''从代码片段集中由 Jim Walsh 编写的公共域 C 程序翻译过来'''
    roman = ""
    n, base10_integer = divmod(base10_integer, 1000)
    roman = "M"*n
    if base10_integer >= 900:
        roman = roman + "CM"
        base10_integer = base10_integer - 900
    while base10_integer >= 500:
        roman = roman + "D"
        base10_integer = base10_integer - 500
    if base10_integer >= 400:
        roman = roman + "CD"
        base10_integer = base10_integer - 400
    while base10_integer >= 100:
        roman = roman + "C"
        base10_integer = base10_integer - 100
    if base10_integer >= 90:
        roman = roman + "XC"
        base10_integer = base10_integer - 90
    while base10_integer >= 50:
        roman = roman + "L"
        base10_integer = base10_integer - 50
    if base10_integer >= 40:
        roman = roman + "XL"
        base10_integer = base10_integer - 40
    while base10_integer >= 10:
```

```

    roman = roman + "X"
    base10_integer = base10_integer - 10
if base10_integer >= 9:
    roman = roman + "IX"
    base10_integer = base10_integer - 9
while base10_integer >= 5:
    roman = roman + "V"
    base10_integer = base10_integer - 5
if base10_integer >= 4:
    roman = roman + "IV"
    base10_integer = base10_integer - 4
while base10_integer > 0:
    roman = roman + "I"
    base10_integer = base10_integer - 1
return roman

```

```
if __name__ == "__main__":
```

'''我们通过十进制整数 `n` 和对罗马数字之间的来回转换来测试转换子程序。如果换作不是等同转换，会产生错误。'''

```

largest_number = 5000
for num in xrange(1, largest_number+1):
    str = DecimalToRomanNumerals(int(num))
    number = RomanNumeralsToDecimal(str)
    if number != num:
        print "Routines failed for", num
        raise "Test failure"
print "Test passed."s

```

它怎样工作

好家伙，你认为像零和小数点这样的简单发明是显而易见地直观，但显然它们不是。在罗马帝国没落之后，它花费了西方文明一千年的时间来改进 Roman 系统。

这个脚本，像本书中的那么多脚本一样，利用 `re` 模块的编译字符串对象搜索和匹配。事实上，用 `RomanNumeralsToDecimal()` 从罗马体到十进制的转换只是穿过输入字符串的搜索过程。它的对应物，`DecimalToRomanNumerals()`，包含稍微较多代码。在执行了一个简单

的 `divmod()` 函数操作之后，广泛地利用 `if` 语句和 `while` 循环把十进制分解为基于十进制的罗马等价物的 11 种类型。`divmod()` 操作指令指出 Python 从多重输出操作中立刻分配两个变量的能力。在这一行中：

```
n, base10_integer = divmod(base10_integer, 1000)
```

变量 `n` 得到了基准数，并且 `base10_integer` 得到了模数。其他的数字用不同的 `if` 测试进行解码。

匿名捐献者

8.4 stats.py

这个脚本可以作为模块使用。统计函数没有关于 Python 的标准。要注意 `stat` 模块没有统计函数，而不管它的名字是什么。

这个脚本是一个和两个向量问题的分析。

```
'''
```

简单统计计算模块

```
'''
```

```
import math
```

```
def XYStats(vector_x, vector_y):
```

```
    if len(vector_x) != len(vector_y):
```

```
        raise "Bad data", "vectors different lengths"
```

```
    results = {
```

```
        "num"      : 0,
```

```
        "meanx"    : 0.0,
```

```
        "meany"    : 0.0,
```

```
        "stddevx"  : 0.0,
```

```
        "stddevy"  : 0.0,
```

```
        "sumx"     : 0.0,
```

```
        "sumxx"    : 0.0,
```

```
        "sumy"     : 0.0,
```

```
        "sumyy"    : 0.0,
```

```
        "sumxy"    : 0.0,
```

```
        "minx"     : 0.0,
```

```
        "maxx"     : 0.0,
```

```
"miny"      : 0.0,
"maxy"      : 0.0
}
minx = 3e308
miny = 3e308
maxx = -minx
maxy = -miny
sx  = 0
sy  = 0
sxx = 0
syy = 0
sxy = 0
for ix in range(len(vector_x)):
    x = vector_x[ix]
    y = vector_y[ix]
    sx = sx + x
    sy = sy + y
    sxx = sxx + x*x
    syy = syy + y*y
    sxy = sxy + x*y
    if x < minx: minx = x
    if x > maxx: maxx = x
    if y < miny: miny = y
    if y > maxy: maxy = y
n = len(vector_x)
meanx = sx/n
meany = sy/n
results["num"]      = n
results["meanx"]    = meanx
results["meany"]    = meany
results["stddevx"]  = math.sqrt((sxx - n*meanx*meanx) / (n - 1))
results["stddevy"]  = math.sqrt((syy - n*meany*meany) / (n - 1))
results["sumx"]     = sx
results["sumxx"]    = sxx
results["sumy"]     = sy
```

```
results['sumyy']    = syy
results['sumxy']    = sxy
results['minx']     = minx
results['miny']     = miny
results['maxx']     = maxx
results['maxy']     = maxy
results['medianx']  = GetMedian(vector_x)
results['mediany']  = GetMedian(vector_y)
return results
```

```
def XStats(vector):
    results = {}
    if len(vector) < 1:
        raise "Bad data", "Null vector"
    min = 3e308
    max = -min
    sx = 0.0
    sxx = 0.0
    for value in vector:
        sx = sx + value
        sxx = sxx + value*value
        if value < min: min = value
        if value > max: max = value
    n = len(vector)
    mean = sx / n
    if n == 1:
        results["stddev"] = 0.0
    else:
        results["stddev"] = math.sqrt((sxx - n*mean*mean) / (n - 1))
    results['mean']    = mean
    results["sum"]     = sx
    results["num"]     = n
    results["min"]     = min
    results["max"]     = max
    results["median"]  = GetMedian(vector)
```

```
    return results

def GetMedian(vector):
    n = len(vector)
    if n < 1:
        raise "Bad data", "Null vector"
    tmp = vector
    tmp.sort()
    if (n+1)/2 == n/2:
        median = (vector[n/2 - 1] + vector[n/2]) / 2.0
    else:
        median = vector[(n+1)/2 - 1]
    return median * 1.0

def Test():
    a=[1,2,3.0]
    r = XStats(a)
    if r["num"] != 3 : raise "TestFailed"
    if r["mean"] != 2.0: raise "TestFailed"
    if r["stddev"] != 1.0: raise "TestFailed"
    if r["sum"] != 6.0: raise "TestFailed"
    if r["min"] != 1.0: raise "TestFailed"
    if r["max"] != 3.0: raise "TestFailed"
    if r["median"] != 2.0: raise "TestFailed"

    s = XYStats(a,a)
    if r["num"] != 3 : raise "TestFailed"
    if s["meanx"] != 2.0: raise "TestFailed"
    if s["meany"] != 2.0: raise "TestFailed"
    if s["stddevx"] != 1.0: raise "TestFailed"
    if s["stddevy"] != 1.0: raise "TestFailed"
    if s["sumx"] != 6.0: raise "TestFailed"
    if s["sumy"] != 6.0: raise "TestFailed"
    if s["sumxx"] != 14.0: raise "TestFailed"
    if s["sumyy"] != 14.0: raise "TestFailed"
```

```
if s["minx"]    != 1.0:  raise "TestFailed"
if s["miny"]    != 1.0:  raise "TestFailed"
if s["maxx"]    != 3.0:  raise "TestFailed"
if s["maxy"]    != 3.0:  raise "TestFailed"
if s["sumxy"]   != 14.0:  raise "TestFailed"

if s["num"]     != 3 :    raise "TestFailed"
raise "Error", "Need to add in correlation/regression stuff"

if __name__ == '__main__':
    Test()
```

它怎样工作

正如本书的其他脚本一样，这个脚本在内部有一个测试序列。如果你从命令行调用这个脚本，它就启动测试序列。虽然它不返回“全清零”；但确认也是否定的。

这个脚本可以扩展为作为一个模块使用。不管它是一个明显的实用工具，这个脚本不使用任何外来的技巧。大部分代码是过时的 Python。

这个脚本采用一个或两个数表作为参数，然后对它们执行基本的统计分析：平均数、中值等。元组不起作用，因为列表必须排序。同样，如果你正在传递 X 和 Y 列表，它们必须具有同样的长度。

注意 XYStats() 中字典的使用。字典被指定为结果，并且已经初始化为所有的关键字和零。稍后，不同的关键字将通过像下面这样的语句用实际数值匹配。

```
results["num"] = n
```

匿名者提供

第 9 章 服务器

本章要点:

- basic.py
- form.py
- helloworld.py
- counter.py
- sengine.py
- engine.py
- who-owns.py
- server.py

Python 非常擅长服务器方面的编程。你能编写 Python 脚本做几乎可以想象的任何事，包括数据库访问。

Internet 对于 Python 来说不成问题。标准 Python 发行版本与分析 XML 和 SGML、编码 HTML 格式和处理 POP3、MIME 的模块以及其他常见的 Internet 规定和协议一起出现，你可以从 www.python.org 下载许多新的模块，这些模块被更新并且比原始的模块更强大。还有一个来自 Digital Creations (www.digicool.com) 的公布环境的称为 Zope 的对象。Zope 是用 Python 编写的具有 C 扩展名。Zope 也位于附带的 CD-ROM 上。

Python 在客户端方面的编程也不低劣。附带的 CD-ROM 上提供的是 Grail，完全用 Python 构建的浏览器。它确实不是 Microsoft 的 Internet Explorer 的竞争者，但它是免费的而且是你自己定制浏览器应用程序的完全基础。它也能像 applet 一样使用 Python 脚本。

按照现实标准来看，这的脚本相当小。真正的服务器方面的脚本通常使用许多模块和数据文件。

9.1 basic.py

这个和它在 CGI（公共网关接口）中得到的一样简单。哦，你能停止 HTML 行“This is a test”，但你不知道你的脚本是否在工作，对吗？

```
#!/usr/bin/python
#这个脚本没有任何实际价值，但它是可想象的最基本的 CGI 脚本
print "Content-type: text/html"
```

```
print #该空白行标明 HTML 标头的结束
print "<p>This is a test</p>"
```

HTML 调用 basic.py:

```
<HTML>
<BODY>
<A HREF "/cgi-bin/basic.py">Click me!</A>
</BODY>
</HTML>
```

它怎样工作

对它来说没有太多的东西。HTML 页要求脚本正确输出 HREF。这个脚本在服务器（我的情况是具有 Apache 的 Caldera Linux 框）上的 cgi-bin 子目录中，并且可以从 Apache 得到这条命令。在这一点上有两个必须遵守的规则。第一是必须用标头标识回答你打算发送的数据类型。既然这样，它就是 text/html。然后需要一个空白行，由单独的 print 语句提供。在那之后打印出来的别的东西作为 HTML 送回客户机。但要记住，你需要给打印行添加 HTML 标签，因为基本的 Python 不能自动生成它们。

9.2 form.py

适用于 CGI 的最普通的用途之一是取得表单的内容并随意摆弄它。这个脚本和 HTML 页组合显示了操作表单数据的一个方法（只是许多方法之一）。

```
#!/usr/bin/python
# form.py
# 解释表单数据的 CGI 脚本

import cgi
import shelve

def memberform()
    list=( )
# 写头部
    print "Content-type: text/html"
    print
# 从表单中获得字段
    form1=cgi.FieldStorage()
```

将字段解码为变量

```
first=form1["first"].value
```

```
last=form1["last"].value
```

```
email=form1["email"].value
```

```
fan=form1["fan"].value
```

向字典中写入字段变量

```
list["first"]=first
```

```
list["last"]=last
```

```
list["email"]=email
```

```
list["fan"]=fan
```

打开 shelve 对象

```
member_list=shelve.open("memlist1")
```

向成员列表中附加新的列表

```
member_list[email]=list
```

打印确认

```
print "<H1> Confirmation!</H1>"
```

```
print "<P> You're entered on the list. Congratulations! </P>"
```

现在关闭 shelve 对象

```
member_list.close()
```

```
if __name__ == "__main__":
```

```
    memberform()
```

HTML 代码:

```
<HTML>
```

```
<BODY>
```

```
<H1>Want to be on our mailing list?</H1>
```

```
<FORM ACTION="/cgi-bin/form.py" METHOD=post>
```

```
First name: <INPUT TYPE=TEXT NAME="first"><BR>
```

```
Last name: <INPUT TYPE=TEXT NAME="last"><BR>
```

```
Email: <INPUT TYPE=TEXT NAME="email"><BR>
```

```
I am a Monty Python fan (highly suggested): <INPUT TYPE=CHECKBOX NAME="fan">
```

```
Click to submit: <INPUT TYPE=SUBMIT VALUE=" Add me! " NAME="submit">
```

```
</FORM>
```

```
</BODY>
```

```
</HTML>
```

它怎样工作

这是一个 CGI 脚本，和像 Astrodome 那样的露营用小帐篷一样简单，但你却能得到其思想。form.py 从客户机上获得表单数据，并把它保存在文件中。它实际上没有用数据做任何事，但至少以后它会被捕捉到。

当服务器从 HTML 页接收提交时脚本就会启动。你能够（而且应该）有更加精心的 HTML 页。而且你能在页面上添加脚本来做错误检验等等这类的事。检测 CGI 脚本保持的方式没有任何错误。例如，它应该检测 e-mail 地址和数码缺席的有效结构或名字中的标点。

这个脚本首先打印 HTML 头标的标准行：

```
print "Content-type: text/html"
print
```

然后，它在变量 form1 中保留不同的域。这些域从 cgi 模块的 FormStorage() 类的实例中提取以下行：

```
form1=cgi.FieldStorage()
```

完整地做这项工作。form1 成为类似字典的对象。正如其他字典一样，它有关键字和数值。关键字是来自表单的字段名，并且数值是那些字段的内容。关于下面这些行：

```
first=form1["first"].value
last=form1["last"].value
email=form1["email"].value
fan=form1["fan"].value
```

这 4 个字段拨出自己的数值并做为变量保存。注意括号内的关键字是表单的字段的名字的字符串文字。

以下行：

```
list["first"]=first
list["last"]=last
list["email"]=mail
list["fan"]=fan
```

创建一个新的字典。用户的第一个名字用关键字“first”成对，最后的名字用关键字“last”等等。

好的，所以现在有一个提取数据的字典。你怎样索引或查找它？下面的行提供了答案：

```
member_list=shelve.open("memlist1")
member_list[email]=list
```

第一行打开一个 shelf 对象。Python 和几个数据库类型模块一起出现，其中的一些适用于真实的数据库，另一些适用于简单的存储方法。shelf 对象只是生成 Python 持久性对象的可能方法之一。还有 pickle、cpickle、gdbm、dumbdbm、anydbm 和 marshal，可以命名少许

几个这样的对象。shelf 对象本质上是一个字典，从而数值可以是任何 Python 对象。既然如此，它是一个字典。换句话说，这个对象将要成为有字典关键字的字典。

shelf 对象可以在包含 CGI 脚本的任意字典中被打开，所以，如果 shelf 不存在，通常运行在 httpd 中的脚本需要来允许创建它，如果它存在就编写它。

为了唯一地标识每个来访者，我选择使用来访者的 e-mail 地址。姓和名可以相同，但是 e-mail 地址却几乎没有相同的。在行 `member_list[email]=list` 中，关键字变量 email 连接到字典变量列表。现在，如果你想要查看一个用户，通过使用 e-mail 地址，你将返回在字典中存储的所有数据。

最后这些行：

```
print "<H1> Confirmation!</H1>"
print "<P> Your're entered on the list. Congratulations! </P>"
member_list.close()
```

打印 HTML 编码来确认已注册，然后关闭对象 member_list。

有时你想给 CGI 脚本传递隐藏的数值，有用户不了解的一些事情。你也许想发送特定的信息给远程客户，并再次返回这些信息进行处理，以避免在你的服务器上存储它。假使那样，你能通过在<FORM>标志中用下面这样的行，给客户编写表单拥有一种数据“状态”：

```
<INPUT name=whatever value=value type=hidden>
```

在“whatever”字段中的无论什么都将送回 CGI 脚本以供将来操作，而终端用户不用知道它在哪里。

9.3 helloworld.py

这个简单脚本在服务器上回送浏览器给出本地时间的页面。

```
#!/usr/local/bin/python
## helloworld.py#from time import *print "Content-type: text/html"print

print "<HTML>"

print "<Head>"
print "<Title>Hello World</Title>"
print "</Head>"

print "<Body>"
```

```

print "<H1>Hello World !</H1>"
print "<hr>"

print "<P>This is Belgium.<br>Localtime is", ctime( time() ) + ".<P>"

print "</Body>"
print "</HTML>"

```

它怎样工作

它本身很简单。这个脚本以标准标头开始，然后提供了这个页面基本的 HTML 代码。在这 4 页面中，它使用标准的 `time` 模块提供了时间标记。

由 Michel Vanaken 提供

9.4 counter.py

`counter.py` 是 Web 的旧标准的 Python 版本，打击计数器。`counter.py` 使用 `gdbm` 模块跟踪打击数。虽然 `counter.py` 实际构建图像计数器，但它与令人厌烦的文本计数器不同。

```

#!/usr/local/bin/python## CGI 计数器 - 使用了 gdbm 数据库
#从字符串导入 atoi、zfill
import gdbm
import cgi
DIGITS - 4

hex_bytes = {
    [ "0xff,0xff,", "0xff,0xff,", "0xff,0xff,", "0xff,0xff,", "0xff,0xff,", \
    "0xff,0xff,", "0xff,0xff,", "0xff,0xff,", "0xff,0xff,", "0xff,0xff," ],
    [ "0x01,0xc0,", "0x01,0xc0,", "0x01,0xc0,", "0x01,0xc0,", "0x01,0xc0,", \
    "0x01,0xc0,", "0x01,0xc0,", "0x01,0xc0,", "0x01,0xc0,", "0x01,0xc0," ],
    [ "0xfd,0xdf,", "0xfd,0xdf,", "0xfd,0xdf,", "0xfd,0xdf,", "0xfd,0xdf,", \
    "0xfd,0xdf,", "0xfd,0xdf,", "0xfd,0xdf,", "0xfd,0xdf,", "0xfd,0xdf," ],
    [ "0x3d,0xde,", "0x7d,0xde,", "0x1d,0xde,", "0x3d,0xdc,", "0xfd,0xdd,", \
    "0x7d,0xd0,", "0xfd,0xd1,", "0x0d,0xd0,", "0x1d,0xdc,", "0x1d,0xde," ],
    [ "0xdd,0xdd,", "0x1d,0xde,", "0x0d,0xdc,", "0xdd,0xd8,", "0xfd,0xdc,", \
    "0x3d,0xd8,", "0x7d,0xde,", "0x0d,0xd8,", "0xcd,0xd9,", "0xcd,0xd9," ],
    [ "0xcd,0xd9,", "0x7d,0xde,", "0xe5,0xd8,", "0xed,0xd9,", "0x7d,0xdc,", \

```



```

"0xbd,0xdf,", "0x3d,0xdf,", "0xf5,0xdb,", "0xe5,0xd3,", "0xed,0xd9," ],
    [ "0xed,0xdb,", "0x7d,0xde,", "0xf5,0xd9,", "0xfd,0xd9,", "0xbd,0xdc,", \
"0x1d,0xdf,", "0x9d,0xdf,", "0xfd,0xdb,", "0xe5,0xd3,", "0xe5,0xd3," ],
    [ "0xe5,0xd3,", "0x7d,0xde,", "0xfd,0xd9,", "0xfd,0xdd,", "0xdd,0xdc,", \
"0x1d,0xdc,", "0xcd,0xdf,", "0xfd,0xdd,", "0xc5,0xd3,", "0xe5,0xd3," ],
    [ "0xe5,0xd3,", "0x7d,0xde,", "0xfd,0xd9,", "0x7d,0xde,", "0xdd,0xdc,", \
"0x7d,0xd8,", "0x2d,0xdc,", "0xfd,0xdd,", "0x8d,0xd9,", "0xe5,0xd3," ],
    [ "0xe5,0xd3,", "0x7d,0xde,", "0xfd,0xdd,", "0x3d,0xdc,", "0xed,0xdc,", \
"0xfd,0xd1,", "0xc5,0xd9,", "0xfd,0xdd,", "0x1d,0xde,", "0xc5,0xd3," ],
    [ "0xe5,0xd3,", "0x7d,0xde,", "0xfd,0xde,", "0xfd,0xd8,", "0xf5,0xdc,", \
"0xfd,0xd3,", "0xe5,0xd1,", "0xfd,0xde,", "0x3d,0xdc,", "0xcd,0xd3," ],
    [ "0xe5,0xd3,", "0x7d,0xde,", "0xfd,0xde,", "0xfd,0xd1,", "0xf5,0xdc,", \
"0xfd,0xd3,", "0xe5,0xd3,", "0xfd,0xde,", "0xcd,0xd8,", "0x1d,0xd8," ],
    [ "0xe5,0xd3,", "0x7d,0xde,", "0x7d,0xdf,", "0xfd,0xd3,", "0x05,0xd0,", \
"0xfd,0xd7,", "0xe5,0xd3,", "0xfd,0xde,", "0xe5,0xd1,", "0xfd,0xd9," ],
    [ "0xed,0xdb,", "0x7d,0xde,", "0xbd,0xdf,", "0xfd,0xd3,", "0xfd,0xdc,", \
"0xfd,0xd7,", "0xe5,0xd3,", "0x7d,0xdf,", "0xe5,0xd3,", "0xfd,0xdd," ],
    [ "0xcd,0xd9,", "0x7d,0xde,", "0xdd,0xd7,", "0xfd,0xdb,", "0xfd,0xdc,", \
"0xfd,0xdb,", "0xcd,0xdb,", "0x7d,0xdf,", "0xe5,0xd3,", "0x7d,0xde," ],
    [ "0xdd,0xdd,", "0x7d,0xde,", "0x0d,0xd0,", "0xcd,0xd9,", "0xfd,0xdc,", \
"0xcd,0xdd,", "0xcd,0xd9,", "0x7d,0xdf,", "0xcd,0xd9,", "0x3d,0xdf," ],
    [ "0x3d,0xde,", "0x1d,0xd8,", "0x05,0xd8,", "0x0d,0xde,", "0xfd,0xdc,", \
"0x0d,0xde,", "0x3d,0xdc,", "0xbd,0xdf,", "0x1d,0xdc,", "0xc5,0xdf," ],
    [ "0xfd,0xdf,", "0xfd,0xdf,", "0xfd,0xdf,", "0xfd,0xdf,", "0xfd,0xdf,", \
"0xfd,0xdf,", "0xfd,0xdf,", "0xfd,0xdf,", "0xfd,0xdf,", "0xfd,0xdf," ],
    [ "0x01,0xc0,", "0x01,0xc0,", "0x01,0xc0,", "0x01,0xc0,", "0x01,0xc0,", \
"0x01,0xc0,", "0x01,0xc0,", "0x01,0xc0,", "0x01,0xc0,", "0x01,0xc0," ],
    [ "0xff,0xff,", "0xff,0xff,", "0xff,0xff,", "0xff,0xff,", "0xff,0xff,", \
"0xff,0xff,", "0xff,0xff,", "0xff,0xff,", "0xff,0xff,", "0xff,0xff," ]
}

```

```
def print_header() :
```

```
#####
```

```
print "Content-type: image/x-bitmap"
```

```

    print
    print "#define counter_width", DIGITS * 16"
    print "#define counter_height 20"
    print "static char counter_bits[] = {"

def print_footer() :
    #####
    print "0x00 ) ;"
    print

def print_digits_values( s ) :
    #####
    i = 0
    while i < 20 :
        for d in s :
            print hex_bytes[ i ][ atoi( d ) ],
        print
        i = i + 1

def inc_counter( s ) :
    #####
    val = atoi( s ) + 1
    return zfill( str( val ), DIGITS )

def get_put_counter( url ) :
    #####
    db = gdbm.open( "counters.gdbm", "w", 0644 )
    if db.has_key( url ) :
        s = db[ url ]
    else :
        s = zfill( '0', DIGITS )

```

```

s = inc_counter( s )
db[ url ] = s
return s

```

```

def CGImain() :
    #####
    list = cgi.SvFormContentDict()
    if list.has_key( "url" ) :
        url = list[ "url" ]
        counter = get_put_counter( url )
        print_header()
        print_digits_values( counter )
        print_footer()

```

```
CGImain()
```

带有嵌入式计数器的典型 HTML 页看起来有点像：

```

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head><title>Counter Example</title></head>
  <body>
    <h1>Counter Example<h1>
    <p>This page has been visited <IMG SRC="/cgi-bin/counter.py">
times.</p>
  </body>
</html>

```

它怎样工作

这个浏览器不直接显示 XBM，它实际上是个图形计数器，但它能像十六进制表示法一样显示装载的图形。那就是为什么有一个巨大的十六进制数值列表。程序提供者说图形绘制很轻便。还说最初像 XBM 一样绘制图形，然后把它应用在 emacs 中得到了它本身正确的十六进制表示。

当模块 cgi 拥有方法 SvFormContentDict()时，这个脚本是为 Python 的旧版本编写的。模块还具有那个方法，并且具有向后兼容性。它现在返回支持 FormStorage。

`SvFormContentDict()`把字段数值存储为字典。在函数 `CGImain()`中，数值字典被检测，查看它是否有 URL 关键字。如果它有，就会调用 `get_put_counter()`，并组合成计数器图形。

`counter.py` 也使用数据库模块 `gdbm`。行：

```
db = gdbm.open( "counters.gdbm", "w", 0644 )
```

使文件 `counters.gdbm` 打开并进行编写。你应该在运行 `counter.py` 之前创建这个文件，虽然如果没有这样做，也会为你创建这个文件。通过把当前目录改为 `cgi-bin` 来创建这个文件，在交互模式下运行 Python，并键入下面的命令：

```
import gdbm
```

```
gdbm.open("counters.gdbm", "n")
```

这样就完成了，你能退出 Python。

好的，现在你已经看过了上面的部分，这还有一些可能出现的问题。

当用这种方法创建数据库时，你需要确信这个数据库是运行脚本的 HTTPD 后台程序可读写的。这样也许有时会有疑问。一个比较可靠的修补程序是：

```
gdbm.open("counters.gdbm", "n")
```

```
import os
```

```
os.chmod("counters.gdbm", 0666)
```

而且，Web 技术发展得如些快速以致于像这样的脚本很快就会过期。既然这样，对于为旧的 `cgi.py` 模块编写的脚本，这样一些修补程序也许是很适用的。例如，你也许使用这个脚本只想发现总是得到同样的计数值，有一些否定了整个组织的数值。如果发生了这种事，它是因为大多数浏览器不能自动刷新计数器，从而从最后一次被检索的计数器中优先使用高速缓存的图像。

为了阻止这种情况发生，你需要给 `print_header()`添加一行，就在“Content-type”行后面。这样就可以工作了，至少可以在 Netscape Navigator 中工作：

```
print "Expires: 0"
```

这一行将导致浏览器一检索到图像就“终止”它，所以它总是把图像的最新拷贝送回到服务器。

某些浏览器也许需要一个实际的时间值，由 `time.ctime()`产生的那一种。其他浏览器也许仍然忽略支持“Pragma: no-cache”的“Expires:”头标。所以像下面这样也许会工作得更好：

```
import time
```

```
print "Expires: ", time.ctime(time.time())
```

```
print "Pragma: no-cache"
```

由 Michel Vanaken 提供

9.5 sengine.py

你怎样才能同时查询多重搜索引擎？好，大概没有精确到微秒，但已足够近了。

```
#!/usr/local/bin/python
#
# sengine.py - 一次搜索多个搜索引擎
# M. Vanaken - 1998
#
# 要求：
# - 线程
#
# 要做的事：
# - 向绝对 URL 写入相对 URL
# - 调查 Note (1) 中的问题
#
# Note(1):
# 在一些系统看来，urllib 不是线程安全。如果可以锁定访问，则这暗示了每个请求的大多数潜在
# 的过程需要更详尽的调查
#
import traceback
from time import sleep
import string
import thread
import urllib
import cgi
# 参阅 Note(1)
#httplib_lock = thread.allocate_lock()
class SearchEngine :
    def __init__( self ) :
        self.Lock = thread.allocate_lock()
        self.Retrieved = 0
        self.CanPrint = 0
        self.Printed = 0
    def EscapeQueryString( self, s ) :
        s = string.strip( s )
```

```

        s = string.joinfields(      string.splitfields( s, " " ), "+" )
        s = urllib.quote( s, "/+:&?" )
        return s

    def Main( self, qs ) :
        try :
            self.Query = self.MakeQueryString( qs )
# 参阅 Note(1)
            # httplib_lock.acquire()
            self.Answer = urllib.urlopen( self.Query )
# 参阅 Note(1)
            # httplib_lock.release()
            self.Lock.acquire()
            self.Retrieved = 1
            self.Lock.release()
            myTurn = 0
            while not myTurn :
                sleep( 1 )
                self.Lock.acquire()
                if self.CanPrint :
                    myTurn = 1
                    self.Lock.release()
            print "\n<hr>\n<H1 align=center>Results from", self.Name,
                "</H1>\n<hr>"
            print
            self.PrintResult()
            print "\n\n<H2 align=center>End of the results from", self.Name,
                "</H2>\n<hr>"
            self.Answer.close()
            self.Lock.acquire()
            self.Printed = 1
            self.Lock.release()
        except :
            self.Lock.acquire()
            self.Printed = 1
            self.CanPrint = 1

```

```

        self.Retrieved = 1
        self.Lock.release()
        traceback.print_exc()
        # self.Answer.close()

class Excite( SearchEngine ) :
    def __init__( self ) :
        SearchEngine.__init__( self )
        self.Name = "Excite"
        self.Url = "http://www.excite.com/"

    def MakeQueryString( self, s ) :
        return self.Url + "search.gw?trace=a&search=" +
self.EscapeQueryString( s )

    def PrintResult( self ) :
        filter = 0
        for line in self.Answer.readlines() :
            if string.find( line, "Ad Start" ) != -1 :
                filter = 1
            if not filter :
                print line
            if string.find( line, "Ad Stop" ) != -1 :
                filter = 0

class InfoSeek( SearchEngine ) :
    def __init__( self ) :
        SearchEngine.__init__( self )
        self.Name = "InfoSeek"
        self.Url = "http://www.infoseek.com/"

    def MakeQueryString( self, s ) :
        return self.Url + "Titles?qt=" + self.EscapeQueryString( s ) +
"&col=WW&sv=IS&lk=noframes&nh=10"

    def PrintResult( self ) :
        for line in self.Answer.readlines() :
            print line

class EuroSeek( SearchEngine ) :
    def __init__( self ) :
        SearchEngine.__init__( self )

```

```

        self.Name = "EuroSeek"
        self.Url = "http://www.euroseek.net/"
    def MakeQueryString( self, s ) :
        return self.Url + "query?iflang=uk&query=" +
self.EscapeQueryString( s ) + "&domain=world&lang=world"
    def PrintResult( self ) :
        for line in self.Answer.readlines() :
            if string.find( line, "Advertisment!" ) == -1 :
                print line
def Search( s ) :
    qs = s,
    Engines = Excite(), InfoSeek(), EuroSeek()
    print "Starting search on"
    print "<UL>"
    for e in Engines :
        print "<LI>" + e.Name
        thread.start_new_thread( e.Main, qs )
    print "</UL>"
    Finished = 0
    while not Finished :
        sleep( 1 )
        SomeOnePrinting = 0
        for e in Engines :
            e.Lock.acquire()
            if e.Retrieved and e.CanPrint and not e.Printed :
                e.Lock.release()
                SomeOnePrinting = 1
                break
            e.Lock.release()
        if not SomeOnePrinting :
            for e in Engines :
                e.Lock.acquire()
                if e.Retrieved and not e.CanPrint :
                    e.CanPrint = 1
                    e.Lock.release()

```



```

        SomeOnePrinting = 1
        break
        e.Lock.release()
    if not SomeOnePrinting :
        Finished = 1
        for e in Engines :
            e.Lock.acquire()
            if not e.Printed :
                e.Lock.release()
                Finished = 0
                break
        e.Lock.release()

#
# 接收来自表单的查询字符串
#
print "Content-type: text/html"
print
print "<HTML>"
print "<Head><Title>MultiSearch results</Title></Head>"
print "<Body>"
form = cgi.FieldStorage()
if form.has_key( "qs" ) :
    qs = form[ "qs" ].value
    Search( qs )
else :
    print "Your Search is empty !"
print "</Body></HTML>"

```

它怎样工作

这个脚本与本书的其他脚本不同，因为它使用线程。使用线程的其他脚本是这个脚本的变体，`engine.py`。如果你没有遇到过线程，这样考虑它们：进程是同胞，而线程是孪生的 Siamese。进程注意它们自己的资源并不情愿地共享它们。线程本能地共享同样的空间。这使线程立刻加快而且比进程更倾向于受到破坏。因为线程访问同样的变量，对它们来说重写彼此的数值是可能的，结果导致关于程序崩溃的所有简洁操作的代码。为了避免这种情况，线程通常锁住某些变量一段时间。Python 用线程模块处理这些事情。

被查询的每个搜索引擎都给出它自己的 `SearchEngine()` 的子类。每个搜索引擎类有它自己的线程，通过利用标记来调整。这项技术加速了查询进程，因为在移到另一个引擎之前，不用等待在一个引擎上执行的查询。

当然，必须适当控制线程以使它们不会在半空中碰撞。`sengine.py` 用被线程查询的标记来处理这件事。这些标记是 `Retrieved`、`CanPrint` 和 `Printed`。

注意这个脚本不能在所有的平台上工作。提供者注释 `urllib` 不总是线程安全的。

由 michel Vanaken 提供

当你正在测试新的 CGI 脚本时，能在同一台机器上用服务器和客户端测试它们。甚至能在同一台机器上使用标准 TCP/IP 和回送地址 127.0.0.1，来测试 Python 服务器和 Python 客户端。

9.6 engine.py

这是由 Michel Vanaken 编写的基于 `sengine.py` 上的脚本。它是第 6 章中 Tkinter GUI 的核心代码。`engine.py` 校正了 Michel 在他的脚本中注释的一些问题，并添加了 GUI。为了运行全部 `engine.py`，包括 GUI，从本章中拷贝 `engine.py`，并从 Tkinter 中拷贝相应的 Tkinter 脚本。整个包可以在附带的 CD-ROM 上得到，包括极好的自述文件。

```
#!/usr/bin/env python
...
```

这是一个多线程的 www 搜索引擎。

它源自 Michel Vanaken 的 `sengine.py`，从那里窃取了几个 URL，以及每个搜索客户端一个线程的主要思想。

Usage:

```
python engine.py <search string>
```

该脚本假定你将 Netscape 放在了路径中。

1999 Mitchell S. Chapman

```
$Id: engine.py,v 1.1 1999/08/22 17:22:27 mchapman Exp mchapman $
...
```

```
__version__ = '$Revision: 1.1 $'
```

```
import sys, threading, Queue, urllib, string, traceback

class Result:
    """该类保持了搜索引擎查询的结果"""
    def __init__(self, engineName, resultStr, finalStatus="Done"):
        self.engineName = engineName
        self.resultStr = resultStr
        self.finalStatus = finalStatus

class SearchBase(threading.Thread):
    """这是一个基本的搜索引擎类。"""
    def __init__(self, rawQueryString, finishCB):
        """初始化一个新实例, 'rawQueryString'描述了要执行的查询。'finishCB'在保存
        查询结果的地方是 MT 安全排队。"""
        # 通过要考虑使用 self.__class__.__name__ 的不利方面, 但是我会完成它的
        threading.Thread.__init__(self, name=self.__class__.__name__)
        self.rawQueryString = rawQueryString
        assert(finishCB != None) # Should assert that it's callable...
        self.finishCB = finishCB
        self.result = None
        self.cancelled = 0

    def run(self):
        """在自己的线程中开始运行自己"""
        result = None
        try:
            query = self.makeQueryString()

            opener = urllib.FancyURLopener()
            resultF = opener.open(query)

            if self.cancelled:
                result = self.cancelledResult()
            else:
```

```
        result = self.readResult(resultF)
    except:
        #而不是排队等候一个正常的结果，而是异常
        result = self.exceptionResult()

    self.finishCB(result)

def cancel(self):
    """Cancel a query."""
    self.cancelled = 1

def makeQueryString(self):
    """在子类中重载该函数，以将 self.rawQueryString 转换成适合于目标搜索引擎的查询字符串。"""
    return self.rawQueryString

def readResult(self, inf):
    """从自己的搜索引擎中返回结果，并适当地重新格式化。"""
    result = None
    content = []
    while not self.cancelled:
        newContent = inf.read(1024)
        if not newContent:
            break
        content.append(newContent)
    content = string.join(content)

    if self.cancelled:
        result = self.cancelledResult()
    else:
        result = Result(self.getName(),
                        self.formattedContent(content))

    return result
```

```
def cancelledResult(self):
    """返回表明取消的结果"""
    result = Result(self.getName(),
        """<HTML><HEAD>
        <TITLE>Search Cancelled</TITLE>
        </HEAD><BODY>
        Search was cancelled.
        </BODY></HTML>""",
        finalStatus="Cancelled")
    return result

def exceptionResult(self):
    """返回一个结果表明由于异常导致失败。"""
    callstack = apply(traceback.format_exception, sys.exc_info())
    msg = """<HTML><HEAD><TITLE>Search Failed</TITLE></HEAD>
    <BODY>
    <H1>Search Failed for %s</H1>
    Reason:<BR>
    <PRE>%s</PRE>
    </BODY>
    </HTML>""" % (self.getName(), string.join(callstack, ""))

    result = Result(self.getName(), msg, finalStatus="Failed")
    return result

def formattedContent(self, rawContent):
    """在子类中重载该函数，以读取和格式化搜索结果。"""
    return rawContent

def escapeQueryString(self, qs):
    """必要时转义查询字符串以便它能被嵌入 URL 中。"""
    result = string.strip(qs)
    result = string.join(string.split(result), "+")
    # 转义结果中的特殊字符，但是不要引用任何的 /、+、:、&或?
    result = urllib.quote(result, '/+:&?')
```

```
    return result
```

```
class Excite(SearchBase):
```

```
    """该搜索线程街道如何和 w.Excite 的搜索引擎对话。"""
```

```
    def makeQueryString(self):
```

```
        return ("http://www.excite.com/search.gw?trace=a&search=%s" %
                (self.escapeQueryString(self.rawQueryString)))
```

```
    def formattedContents(self, rawContent):
```

```
        """从响应中删除广告"""
```

```
        filter = 0 # 非 0 则表明应忽略当前行
```

```
        result = []
```

```
        lines = string.split(rawContent, "\n")
```

```
        for line in lines:
```

```
            if string.find(line, "Ad Start") != -1:
```

```
                filter = 1
```

```
            elif string.find(line, "Ad Stop") != -1:
```

```
                filter = 0
```

```
            else:
```

```
                if not filter:
```

```
                    result.append(line)
```

```
        return string.join(result, "\n")
```

```
class InfoSeek(SearchBase):
```

```
    def makeQueryString(self):
```

```
        return ("http://www.infoseek.com/Titles?"
```

```
                "qt=%s&col=WW&sv=IS&lk=noframes&nh=10" %
```

```
                (self.escapeQueryString(self.rawQueryString)))
```

```
class EuroSeek(SearchBase):
```

```
    def makeQueryString(self):
```

```
        return ("http://www.euroseek.net/query?iflang=uk&query=%s"
```

```
                "&domain=world&lang=world" %
```

```
(self.escapeQueryString(self.rawQueryString)))

def formattedContents(self, rawContent):
    """从EuroSeek 响应中过滤广告。"""
    result = []
    lines = string.split(rawContent, "\n")
    for line in lines:
        if string.find(line, "Advertisement!") == -1:
            result.append(line)
    return string.join(result, "\n")

class SearchManager:
    """该类管理跨越多个引擎的搜索的执行。"""
    def __init__(self):
        self.queue = Queue.Queue(0)
        self.engines = [Excite, InfoSeek, EuroSeek]

    def query(self, queryStr):
        """执行一个查询。返回 Result 对象列表。"""
        for engine in self.engines:
            thread = engine(queryStr, self.threadDoneCB)
            thread.start()

        results = []
        # 我们应从每个线程中获得确切的结果。
        for i in range(len(self.engines)):
            # 如有必要, 阻止一个线程, 直到下一个结果到达为止。
            results.append(self.queue.get())
        return results

    def threadDoneCB(self, result):
        """当完成一线程时激活回叫信号。这种方法是在完成线程的环境下激活的, 所以必须是重新可以进入的。"""
        self.queue.put(result)
```

```

def main():
    """模块主线（用于独立的运行）"""
    import sys, os, tempfile

    mgr = SearchManager()
    queryStr = "Herbert Hoover"
    if len(sys.argv) > 1:
        queryStr = sys.argv[1]

    # 将结果转储到一组临时文件中，因为这个脚本不能去除 '<HTML>' 和来自检索结果的链接，将
    # 每个搜索结果保存到临时文件中，然后创建一个可以链接到它们上的主索引。
    basename = tempfile.mktemp()
    indexname = "%s.html" % basename
    outf = open(indexname, "w")
    print "Storing results in", indexname

    outf.write("""<HTML>
<HEAD><TITLE>Search Results For %s</TITLE></HEAD>
<BODY>
<H2>Search Results For %s</H2>
<UL>
    """ % (queryStr, queryStr))

    results = mgr.query(queryStr)
    for result in results:
        resultFilename = "%s_%s.html" % (basename, result.engineName)
        resultF = open(resultFilename, "w")
        resultF.write(result.resultStr)
        resultF.close()
        outf.write('<LI><A HREF="file:%s">%s</A>\n' %
                    (resultFilename, result.engineName))
    outf.write("""</UL></BODY></HTML>""")
    outf.close()

```



```
# 假设在 Linux 上运行的用户安装了最新的 Netscape 版本，那么在临时文件中可以指出它们
# 的浏览器：

cmd = "netscape --remote 'openURL(file:%s)'" % indexname
status = os.system(cmd)

if status:
    print "Attempt failed w. status %d." % status

if __name__ == "__main__":
    main()
```

它怎样工作

要得到更多的详细资料，请查看 CD-ROM 上的 `engine.zip` 中的自述文件。提供者已经解释了他正在做的事情，比我讲的要好得多。

由 Mitch Chapman 提供

9.7 who-owns.py

这个特定的 Linux 脚本调查谁正在指定的套接字处监听。提供者注释每次出现新的 Linux 内核时，这个脚本似乎像失败了一样。但 `who-owns.py` 能生成一个优秀的启动软件包，从而得到一个更稳定的版本。

```
# -*- Mode: Python; tab-width: 4 -*-

import os
import regex
import string
import sys

# 与 linux 有关的
# 弄清什么进程正在一个特定套接字上监听。

def inode_of_socket (port, type='tcp', server_only=1):
    target_port = string.upper ('%04x' % (string.atoi (port)))
    lines = open ('/proc/net/%s' % type).readlines()
    inode = 0
    for line in lines[1:]:
```

```
fields = string.split (line)
[addr, port] = string.split (fields[1], ':')
if port == target_port:
    if (not server_only) or (addr == '00000000'):
        inode = string.atoi (fields[9])
        break
return inode

pid_regex = regex.compile ('\\([0-9]+\\)')

def process_of_inode (inode):
    #inode = '[0000]:%d' % inode
    inode = 'socket:[%d]' % inode
    cwd = os.getcwd()
    try:
        pids = filter (
            lambda x: pid_regex.match (x) == len(x),
            os.listdir('/proc')
        )
        pids.remove (str(os.getpid()))
        for pid in pids:
            fd_dir = '/proc/%s/fd' % pid
            if os.path.isdir (fd_dir):
                os.chdir (fd_dir)
                links = map (
                    os.readlink,
                    os.listdir ('.')
                )
                if inode in links:
                    return pid
    finally:
        os.chdir (cwd)
    return 0

if __name__ == '__main__':
```

```

if len(sys.argv) < 2:
    print 'Usage: %s port [tcp|udp]' % sys.argv[0]
else:
    if len(sys.argv) < 3:
        socket_type='tcp'
    else:
        socket_type=sys.argv[2]
    port = sys.argv[1]
    inode = inode_of_socket (port, socket_type)
    if not inode:
        print "Couldn't find inode for socket on %s port %s" % (
            socket_type,
            port
        )
        sys.exit(0)
    else:
        print 'inode: %d' % inode
        process = process_of_inode (inode)
        if not process:
            print "Couldn't find process for inode %d" % inode
        else:
            print open('/proc/%s/status' % process).read()

```

它怎样工作

`who-owns.py` 监控一个特别的套接字并决定哪个进程为监听那个套接字负责。如果你怀疑电脑黑客正在你的机器上运行可疑的程序，用这个脚本处理是很方便的。

意识到正如提供者注释的那样，这个脚本可以和每个新的 Linux 内核紧密联系。可见，它现在在 2.0.36 上工作。

`inode_of_socket()` 能算出你感兴趣的套接字的索引节。它在伪文件 `/proc/net/tcp` 或 `/proc/net/udp` 中读取数据，并得到套接字绑定的 IP 地址和端口号。

然后，`process_of_inode()` 会发现拥有索引节的进程。注意在某些 Linux 系统上，你也许必须注释掉下面的行：

```
inode='socket:[%d]' % inode
```

不注释掉行：

```
inode='[0000]:%d' % inode
```

进行这项工作。零是必要的，因为 Linux 习惯给所有的套接字设备号 0000。

`process_of_inode()` 现在从列表中删除它自己的 PID（进程标识符），因为毕竟它知道它不是黑客进程。

`process_of_inode()` 现在有几个文件要处理。它必须在 `/proc/` 中访问设备文件，寻找与 `inode-process` 相匹配的进程。如果它发现有一个匹配，它就复制关于该进程的信息。如果它没有发现匹配，`process_of_inode()` 返回零。

由 Sam Rushing 提供

9.8 server.py

```
# 一个简单的"chat"服务器 在端口 4000 上创建一个服务器
# 用户从端口 4000 远程登录到你的计算机上，并且彼此间可以聊天，使用"quit"来断开连接，
# 使用"shutdown"来关闭服务器。要求用套接字。
#
# 7/31/96 J. Strout      http://www.strout.net/

# 导入所需模块：

from socket import *      # 获取套接字，为安全起见，使用套接字
import string             # 字符串函数
import time               # 用于 sleep(1) 函数

# 定义全局变量

HOST = ''                 # 表示本地主机的符号名称
PORT = 4000               # 任意的没有特权的服务器
endl = '\r\n'             # 标准的终止行结尾

userlist = []             # 连接的用户列表
done = 0                  # 设置为 1 以关闭它

kAskName = 0              # 用于标记的一些常量
kWaitName = 1             # 每位用户的状态
kOK = 2
```

存储关于连接的用户的信息的类

```
class User:
    def __init__(self):
        self.name = ""
        self.addr = ""
        self.conn = None
        self.step = kAskName

    def Idle(self):
        if self.step == kAskName: self.AskName()

    def AskName(self):
        self.conn.send("Name? ")
        self.step = kWaitName

    def HandleMsg(self, msg):
        print "Handling message: ",msg
        global userlist

        #如果等待字符名称, 则记录它
        if self.step == kWaitName:
            #试着捕捉由一些远程登录发送的初始信息。
            if len(msg) < 2 or msg=="#": return
            print "Setting name to: ",msg
            self.name = msg
            self.step = kOK
            self.conn.send("Hello, "+self.name+endl)
            broadcast(self.name+" has connected."+endl)
            return

        # 检查命令
        if msg == "quit":
            broadcast(self.name+" has quit.\n")
```

```
        self.conn.close()
        userlist.remove(self)
        return
```

```
    #否则, 广播msg
```

```
    broadcast( self.name+": "+msg+endl )
```

```
# 检查进入连接的程序
```

```
def pollNewConn():
    try:
        conn, addr = s.accept()
    except:
        return None
    print "Connection from", addr
    conn.setblocking(0)
    user = User();
    user.conn = conn
    user.addr = addr
    return user
```

```
# 向所有连接的用户广播消息的程序
```

```
def broadcast(msg):
    for u in userlist:
        u.conn.send(msg)
```

```
# 主程序
```

```
# 建立服务器
```

```
s = socket(AF_INET, SOCK_STREAM)
s.bind(HOST, PORT)
s.setblocking(0)
s.listen(1)
print "Waiting for connection(s)..."

#一直循环，直到完成为止，处理连接和进入的消息

while not done:
    time.sleep(1)          # 进入睡眠状态以减少处理器的使用量
    u = pollNewConn()      # 检查进入的连接
    if u:
        userlist.append(u)
        print len(userlist), "connection(s)"

    for u in userlist: # 检查所有连接的用户
        u.Idle()
        try:
            data = u.conn.recv(1024)
            data = filter(lambda x: x>=' ' and x<='z', data)
            data = string.strip(data)
            if data:
                print "From",u.name,': [' +data+']'
                u.HandleMsg(data)
                if data == 'shutdown': done=1
        except:
            pass

    for u in userlist:
        u.conn.close()
```

它怎样工作

使用这个脚本你能让其他的远程登录在端口 4000 连到你的机器上，并进行一个聊天会话。甚至当有人登录时，这个脚本还会通知用户。在我的印象中，server.py 只是 Tkinter GUI 的表示，但怎样使用那可是你的事情。

主程序设置服务器——打开套接字，绑定它等。然后，这个脚本进入无穷循环等待其他人登录。每个新参与者都有用 `User` 类存储的自己的信息。

`pollNewConn()`不断地检查新的连接，而且当它发现一个时，就实例化 `User()`。

由 Joe Strout 提供，他是 Salk Institute in La Jolla, CA 的科学软件开发者

第 10 章 字符串和其他数据类型

本章要点:

- strplay.py
- dog.py
- banner.py
- ebcdic.py
- lc.py
- sample.py
- xref.py
- piglatin.py

啊，字符串。Python 可以用字符串做伟大的工作。你能切片、重新整理、链接和做大量其他有趣的事情，而且它非常容易。

10.1 strplay.py

这个脚本没有什么值得使用，但它说明了几个基本字符串函数。它能很容易地被修改，例如，打开一个文件并搜索单词或字母。要运行 strplay.py，把它拷贝到正确的 Python 目录下，设置执行权限并键入：

```
python strplay.py
```

然后，按 **Enter** 键。你能看到下面的提示：

```
Type a few words for me, please:
```

键入几个单词并再次按下 **Enter** 键。

```
#!/usr/bin/python
```

```
# 字符串表演
```

```
import string
```

```
def string_play(instr):
```

```
    print 'You typed: ', instr
```

```

ecount=string.count(instring, "e")
print "There are",ecount, "instances of e in this sample."
upstring=string.upper(instring)
dnstring=string.lower(instring)
print "All uppercase:", upstring
print "All lowercase:", dnstring
outstring=string.split(instring)
print "The string breaks into these elements: ",outstring
lenstring=len(outstring)
print "Which number: ", lenstring
if lenstring<3:
    print "I would do a slice, but a I need at least 3 elements."
else:
    sliced=outstring[:2]
    print "The first two elements are:", sliced
outstring.sort()
print "Sorted, it looks like this:",outstring
outstring.reverse()
print "In reverse order:", outstring

if __name__ == "__main__":
    instring=raw_input("Type a few words for me please: ")
    string_play(instring)

```

它怎样工作

这个脚本说明了 Python 中的部分基本字符串处理的概念。脚本以 main 开始执行，简单地写成：

```

if __name__ == "__main__":
    instring=raw_inpjt("Type a few word for me please: ")
    string_play(instring)

```

当脚本被期待作为主程序而不是作为模块执行时，使用“if”语句。在交互模式中你也能通过启动解释程序、输入 `strplay` 并执行下面所示的行，来运行这个脚本：

```
>>>strplay.string_play("I'll have your Span")
```

脚本中发生的下一件事是 `raw_input()` 语句的执行。这个语句会在屏幕上放置一个提示符，从标准输入设备（在这这是键盘）中取得输入，并且装载具有字符串数值的变量 `instring`。

数值可以是任意长度。在用户键入一些单词并按下 Enter 键后, `string_play()` 就会被调用, 传递 `instring`。

函数 `string_play()` 有全部的处理代码。脚本输入 `string` 模块所以 `string` 的方法是可用的。前面几行是简单的字符串处理函数。

第一行按你最初键入它的方式正确地打印它。

第二行检查字母“e”的存在, 并计算有多少个实例。然后, 结果在第三行打印出来。

第四和第五行使用两个 `string` 模块的方法: `upper()` 和 `lower()`。

第六行和第七行打印大写字母盘和小写字母盘转换的结果。

```
print "You typed: ", instring
ecount=string.count(instring, "e")
print "There are", ecount, "instances of e in this sample."
upstring=string.upper(instring)
dnstring=string.lower(instring)
print "All uppercase:", upstring
print "All lowercase:", dnstring
```

随后的行有些细节上的不同。第一行:

```
outstring=string.split(instring)
```

仍使用这个 `string` 模块的另一个方法, 把字符串转换为列表, 从而改变数据的整个结构。现在, 你键入的每个单词都成为一个列表元素。

这样做的意义是你已经从字符串函数到列表函数交叉了行。当你真正拥有一个列表时, 忘记数据类型已经改变和让编写的代码用字符串工作是容易的。下一行用图形方式论证了这一点, 通过打印列表:

```
print "The string breaks into these elements: ", outstring
```

然后, 脚本算出列表有多长:

```
lenstring=len(outstring)
```

于是, 然后打印元素的数目:

```
print "Which number: ", lenstring
```

接下来几行做一些简单的错误检测和切片。切片意味着从序列中拷贝所选择的元素。但是如果你只有一个元素, 就不能切出两个元素。例如, 如果你尝试这么做, Python 不会有任何抱怨, 只是返回一个元素。但是如果程序在寻找两个元素并且只得到一个元素, 你将会得到一个异常。因此出现检查元素数目错误。

所以, 在切片之前, `if` 语句会查看是否至少有三个元素。如果不是至少三个元素, 脚本会显示出警告。如果有三个或多个元素, 就执行 `else` 语句。

`slice` 表达式使用索引。在 Python 中, 每个序列都有一个能直接编址的索引。冒号左边的数字是出发点, 而右边的数字是终点。计数从零开始。变量 `[1:5]` 在从头至尾五个元素的第

二元素切出，就在索引中不到数字 5 的地方停止。

作为一个例子，如果用户键入 “It's a fair cop”，它就被转换为[“It's”，“a”，“fair”，“cop”]。切片的元素在变量 `outstring` 中。通过给出索引[:2]，会告诉 Python 从第一个元素开始并采用每个元素直到第三个元素，得到总数 2。在这个例子中，就成为[“it's”，“a”]。

```
if lenstring<3
    print "I would do a slice, but a I need at least 3 elements."
else:
    sliced=outstring[:2]
    print "The first two elements are:", sliced
```

然后，打印出新缩短的列表。在编入索引的 `slice` 语句上有几个变化。语句 `sequence[:]` 返回所有的序列。`Sequence[1:]`得到从数字 2 到结尾的所有元素，甚至可以使用负数在索引中来回移动。`Sequence[:-1]`给出从第 1 个到结尾前一个的所有元素。

最后几行是：

```
outstring.sort()
print 'Sorted, it looks like this:',outstring
outstring.reverse()
print "In reverse order:", outstring
```

这里，按次序，我已经排序了 `outstring` 的元素，打印了排序表，然后反向排序列表并且再打印一次。用这些单词 “It's a fair cop” 运行这个脚本并查看发生了什么事很有趣。查看排序表，由于某些原因，“It's” 在 “a” 的前面。排序算法首先寻找大写字母，然后是小写字母。如果你想真实地排序，确信所有的元素都相同地用大写字母开头通常是最好的方法。

10.2 dog.py

输入你的年龄，这个脚本会用 `dog` 年历返回你的年龄。

###获得初始年龄

```
age = input("Enter your age (in human years): ")
```

```
print # 打印空白行
```

检查范围，然后打印结果

```
if age < 0:
    print "Negative age?!? I don't think so."
elif age < 3 or age > 110:
    print "Frankly, I don't believe you."
else:
    print "That's", age*7, "in dog years."
```

```
###暂停以等待 Return 键（所有窗口并不消失）
```

```
raw_input('press Return')
```

它怎样工作

你输入年龄；脚本乘以 7 并打印出结果。然而，它也做一些错误检验，确信你没有试图欺骗脚本。

```
if age < 0:
    print "Negative age?!? I don't think so."
elif age < 3 or age > 110:
    print "Frankly, I don't believe you."
else:
    print "That's", age*7, "in dog years."
```

脚本查看你键入的年龄，看看是否为负数、是否在 3 以下、或者超过 110。它分三步进行工作：一个 if 语句、一个 elif 和一个 else。最后，当脚本对你的诚实满意后，它就打印出结果。

由 Joe Strout 提供，他是 California 的 La Jolla 的 Salk Institute 的科学软件开发者。

10.3 banner.py

脚本打印“**banner**（标题）”，即你键入的单词或短语的具有传奇色彩的描写。运行这个脚本的一种容易的方法是在交互模式下启动 Python，输入 banner 并且使用命令：

```
banner.Banner("string", "character")
```

使脚本在标题中使用字符串作为标题文字和使用字符作独立元素。

```
...
```

这个模块包含了可用以打印横幅信息的函数 Banner()，类似于 UNIX banner(1) 函数。。

```
...
```

```
import sys
```

```
...
```

数组字母包含如何打印 32 和 126 之间的每个字符的信息。每个字符有 8 个字节，每个字节代表字体的一行。第一个数字的高字节是 8 位的第一行，下一字节是下一行，依次类推。我通过写一个分析某人的横幅程序输出的脚本得到了这些数字。

```

letters = [
    [ 0x00000000, 0x00000000 ], [ 0x30303030, 0x30003000 ],
    [ 0x6c6c6c00, 0x00000000 ], [ 0x6c6cfe6c, 0xfe6c6c00 ],
    [ 0x187e407e, 0x027e1800 ], [ 0xc2c60c18, 0x3066c600 ],
    [ 0x3828387b, 0xd6cc7700 ], [ 0x60204000, 0x00000000 ],
    [ 0x1c70c0c0, 0xc0701c00 ], [ 0x701c0606, 0x061c7000 ],
    [ 0x006c38fe, 0x386c0000 ], [ 0x303030fc, 0x30303000 ],
    [ 0x00000000, 0x00602040 ], [ 0x0000007c, 0x00000000 ],
    [ 0x00000000, 0x00060600 ], [ 0x02060c18, 0x3060c000 ],
    [ 0x7cc6ced6, 0xe6c67c00 ], [ 0x10703030, 0x30307800 ],
    [ 0x78cc0c18, 0x3062fe00 ], [ 0x78cc0c38, 0x0ccc7800 ],
    [ 0x0c1c6ccc, 0xfe0c1e00 ], [ 0x7e40407c, 0x06c67c00 ],
    [ 0x3c64c0fc, 0xc6c67c00 ], [ 0xfe860c18, 0x18181800 ],
    [ 0x3c66663c, 0x66663c00 ], [ 0x7cc6c67e, 0x064c7800 ],
    [ 0x00006060, 0x00606000 ], [ 0x00006060, 0x00602040 ],
    [ 0x183060c0, 0x60301800 ], [ 0x00007c00, 0x7c000000 ],
    [ 0xc0603018, 0x3060c000 ], [ 0x78cc8c1c, 0x30003000 ],
    [ 0x3c46c2ce, 0xcc407800 ], [ 0x183c6666, 0x7e666600 ],
    [ 0xfc66667c, 0x6666fc00 ], [ 0x3c66c0c0, 0xc2663c00 ],
    [ 0xfc666666, 0x6666fc00 ], [ 0xfe626878, 0x6862fe00 ],
    [ 0xfe626878, 0x6860f000 ], [ 0x3c64c0c0, 0xce663a00 ],
    [ 0xc0ccccfc, 0xc0cccc00 ], [ 0x3c181818, 0x18183c00 ],
    [ 0x3c181818, 0x98d87000 ], [ 0xe6666c78, 0x6c66e600 ],
    [ 0xf0606060, 0x6066fe00 ], [ 0xc6eefed6, 0xc6c6c600 ],
    [ 0xc6e6f6de, 0xc6c6c600 ], [ 0x7cc6c6c6, 0xc6c67c00 ],
    [ 0xfc66667c, 0x6060f000 ], [ 0x7cc6c6c6, 0xc6c67c06 ],
    [ 0xfc66667c, 0x6c66e600 ], [ 0x7ec2c07c, 0x0686fc00 ],
    [ 0x7e5a1818, 0x18183c00 ], [ 0x66666666, 0x66663c00 ],
    [ 0xc6c6c66c, 0x6c381000 ], [ 0xc6c6c6d6, 0xfeec600 ],
    [ 0xee6c3810, 0x386cee00 ], [ 0xc3663c18, 0x18183c00 ],
    [ 0xfe860c18, 0x3062fe00 ], [ 0x7c606060, 0x60607c00 ],
    [ 0xc0603018, 0x0c060200 ], [ 0x3e060606, 0x06063e00 ],
    [ 0x10386cc6, 0x00000000 ], [ 0x00000000, 0x00007c00 ],

```

```
[ 0x0c080400, 0x00000000 ], [ 0x00007c04, 0xfc8cfa00 ],
[ 0xe060607c, 0x6666fc00 ], [ 0x00007cc6, 0xc0c67c00 ],
[ 0x1c0c0c7c, 0xcccc7a00 ], [ 0x00007cc2, 0xfec07c00 ],
[ 0x386c60f8, 0x6060f000 ], [ 0x00007bc6, 0xc67e047c ],
[ 0xe060606e, 0x7666e700 ], [ 0x18003818, 0x18183c00 ],
[ 0x18003c18, 0x1818d870 ], [ 0xe060666c, 0x706ce600 ],
[ 0x38181818, 0x18183c00 ], [ 0x0000ecd6, 0xd6c6e700 ],
[ 0x00006e76, 0x66666600 ], [ 0x00003c66, 0x66663c00 ],
[ 0x0000dc66, 0x667c60f0 ], [ 0x000076cc, 0xcc7c0c1e ],
[ 0x0000dc76, 0x6060f000 ], [ 0x0000fec0, 0xfe06fe00 ],
[ 0x10307c30, 0x30361c00 ], [ 0x0000cecc, 0xcccc7600 ],
[ 0x00006666, 0x663c1800 ], [ 0x0000c6d6, 0xd6fe6c00 ],
[ 0x0000c66c, 0x386cc600 ], [ 0x0000c6c6, 0xc67e0478 ],
[ 0x0000fc98, 0x3064fc00 ], [ 0x0c181830, 0x18180c00 ],
[ 0x10101000, 0x10101000 ], [ 0x60303018, 0x30306000 ],
[ 0x66980000, 0x00000000 ]
]
```

```
def Banner(string, char_to_use):
    global letters
    out = [ [], [], [], [], [], [], [], [] ] # 数据的 8 行
    for ltr in range(len(string)):
        char = string[ltr]
        if ord(char) < 32 or ord(char) > 126:
            char = ' '
        ix = ord(char)-32
        bytes = []
        lines = letters[ix][0]
        #print "Lines = 0x%08x" % lines
        out[0].append(((lines & (0xff << 24)) >> 24) & 0xff)
        out[1].append(((lines & (0xff << 16)) >> 16) & 0xff)
        out[2].append(((lines & (0xff << 8)) >> 8) & 0xff)
        out[3].append(lines & 0xff)
        lines = letters[ix][1]
```

```

    #print "Lines = 0x%08x" % lines
    out[4].append(((lines & (0xff << 24)) >> 24) & 0xff)
    out[5].append(((lines & (0xff << 16)) >> 16) & 0xff)
    out[6].append(((lines & (0xff << 8)) >> 8) & 0xff)
    out[7].append(lines & 0xff)
    *
for element in out:
    for byte in element:
        PrintByteLine(byte, char_to_use)
    sys.stdout.write("\n")
sys.stdout.write("\n")

def PrintByteLine(byte, char_to_use):
    for ix in range(8):
        if byte & (1 << (8 - ix)):
            sys.stdout.write("%s" % char_to_use)
        else:
            sys.stdout.write(" ")

if __name__ == '__main__':
    import sys
    if len(sys.argv) < 2:
        print "Usage: banner <string>"
        sys.exit(1)
    Banner(sys.argv[1], "X")

```

它怎样工作

函数 **Banner**（记住 Python 是区分大小写的，所以不要用“banner”调用它）用两个变量来定义，用于构造标题的字符串和字符。下一行声明了变量 **letters** 为全局变量，但实际上在 Python 的现今版本中它并不是真正必要的。变量 **out** 声明为 8 个列表的一个列表，所有的列表都为空。

```

def Banner(string, char_to_use):
    global letters
    out = [ [], [], [], [], [], [], [], [] ] # 数据的 8 行

```

现在，脚本必须处理字符串并把它转换成图形表示的行，而不是字符串。为了达到这个目的，脚本需要一个显示如何从单个字符中构建字母外观的查找表格。这个字母列表只是一

个查找表格。Python 没有矩阵或数组数据类型，但是一列元素列表可以做许多相同的工作。以下行：

```
if ord(char) < 32 or ord(char) > 126:
    char = " "
```

检查非打印的比 32 低或比 126 高的 ASCII 字符。可印刷的字符存储在字母列表中，所以脚本现在准备看看使用哪些字符。行：

```
ix = ord(char)-32
从 32 到 0 预置索引到字符。行：
out[0].append(((lines & (0xff << 24)) >> 24) & 0xff)
out[1].append(((lines & (0xff << 16)) >> 16) & 0xff)
out[2].append(((lines & (0xff << 8)) >> 8) & 0xff)
out[3].append(lines & 0xff)
```

说明 `append` 方法如何工作。记住变量 `out` 是 8 个列表的一列。从元素零开始，用变量 `out` 中写给 8 个列表中的每一个列表。这给出用于 banner 的一条垂直线。

这些行实际上是被组装的。例如，在第一行中，脚本在变量 `out` 中用括号中的逐位运算的结果附加（或者“替换”，如果你愿意）第一个列表。`0xff` 是十进制 255 的十六进制表示法。注意，虽然这 4 个各自的逐位运算正在前三行的每一行中进行。现在这是比较有效率的。行 `sys.stdout.write("%s" % char_to_use)` 在函数 `PrintByteLine` 中特别有趣。首先，它使用 `sys.stdout.write` 代替普通的打印语句。并且它使用格式化。代码片断 `"%s" % char_to_use` 利用 `"%s"` 作为代替 `char_to_use` 中的字符的占位符。当由 `Banner` 调用 `PrintByteLine` 时，会反复做这项工作。

匿名者提供

10.4 ebcdic.py

很久以前，当 IBM 创建自己的字符集合时，被称为 EBCDIC（扩充的二进制编码的十进制交换码）。ASCII 和 EBCDIC 都是 8 位格式，但 EBCDIC 实际上使用所有的 8 位，而 ASCII 只使用 7 位。EBCDIC 仍然在 IBM Big Iron（大型机）上使用，并且当你看见数据从 Big Blue 大型机上输出时，有许多次它仍然使用这种格式。你也许想要把 EBCDIC 转换为更通用的 ASCII，这就是这个脚本进入角色的地方。`ebcdic.py` 在两个方向进行转换，从 ASCII 到 EBCDIC，反之亦然。

这个脚本意味着可以作为一个模块而不是独立程序。如果你单独运行它，它将测试它自身。

```
...
```

ASCII <=> EBCDIC 转换函数。数组是从 `Snippets` 集合中获得的。

```

a2eG = [
    0, 1, 2, 3, 5, 45, 46, 47, 22, 5, 37, 11, 12, 13, 14, 15,
    16, 17, 18, 19, 60, 61, 50, 38, 24, 25, 63, 39, 28, 29, 30, 31,
    64, 79, 127, 123, 91, 108, 80, 125, 77, 93, 92, 78, 107, 96, 75, 97,
    240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 122, 94, 76, 126, 110, 111,
    124, 193, 194, 195, 196, 197, 198, 199, 200, 201, 209, 210, 211, 212, 213, 214,
    215, 216, 217, 226, 227, 228, 229, 230, 231, 232, 233, 74, 224, 90, 95, 109,
    121, 129, 130, 131, 132, 133, 134, 135, 136, 137, 145, 146, 147, 148, 149, 150,
    151, 152, 153, 162, 163, 164, 165, 166, 167, 168, 169, 192, 106, 208, 161, 7,
    32, 33, 34, 35, 36, 21, 6, 23, 40, 41, 42, 43, 44, 9, 10, 27,
    48, 49, 26, 51, 52, 53, 54, 8, 56, 57, 58, 59, 4, 20, 62, 225,
    65, 66, 67, 68, 69, 70, 71, 72, 73, 81, 82, 83, 84, 85, 86, 87,
    88, 89, 98, 99, 100, 101, 102, 103, 104, 105, 112, 113, 114, 115, 116, 117,
    118, 119, 120, 128, 138, 139, 140, 141, 142, 143, 144, 154, 155, 156, 157, 158,
    159, 160, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183,
    184, 185, 186, 187, 188, 189, 190, 191, 202, 203, 204, 205, 206, 207, 218, 219,
    220, 221, 222, 223, 234, 235, 236, 237, 238, 239, 250, 251, 252, 253, 254, 255
]

```

```

e2aG = [
    0, 1, 2, 3, 156, 9, 134, 127, 151, 141, 142, 11, 12, 13, 14, 15,
    16, 17, 18, 19, 157, 133, 8, 135, 24, 25, 146, 143, 28, 29, 30, 31,
    128, 129, 130, 131, 132, 10, 23, 27, 136, 137, 138, 139, 140, 5, 6, 7,
    144, 145, 22, 147, 148, 149, 150, 4, 152, 153, 154, 155, 20, 21, 158, 26,
    32, 160, 161, 162, 163, 164, 165, 166, 167, 168, 91, 46, 60, 40, 43, 33,
    38, 169, 170, 171, 172, 173, 174, 175, 176, 177, 93, 36, 42, 41, 59, 94,
    45, 47, 178, 179, 180, 181, 182, 183, 184, 185, 124, 44, 37, 95, 62, 63,
    186, 187, 188, 189, 190, 191, 192, 193, 194, 96, 58, 35, 64, 39, 61, 34,
    195, 97, 98, 99, 100, 101, 102, 103, 104, 105, 196, 197, 198, 199, 200, 201,
    202, 106, 107, 108, 109, 110, 111, 112, 113, 114, 203, 204, 205, 206, 207, 208,
    209, 126, 115, 116, 117, 118, 119, 120, 121, 122, 210, 211, 212, 213, 214, 215,
    216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231,
    123, 65, 66, 67, 68, 69, 70, 71, 72, 73, 232, 233, 234, 235, 236, 237,

```

```
125, 74, 75, 76, 77, 78, 79, 80, 81, 82, 238, 239, 240, 241, 242, 243,
92, 159, 83, 84, 85, 86, 87, 88, 89, 90, 244, 245, 246, 247, 248, 249,
48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 250, 251, 252, 253, 254, 255
]

def AsciiToEbcDic(str):
    ''' 返回 EBCDIC 表中的 ASCII 字符串 str. '''
    global a2eG
    if type(str) != type(''):
        raise "Bad data", "Expected a string argument"
    if len(str) == 0: return str
    newstr = ""
    for ix in xrange(len(str)):
        newstr = newstr + chr(a2eG[ord(str[ix])])
    return newstr

def EbcDicToAscii(str):
    global e2aG
    if type(str) != type(''):
        raise "Bad data", "Expected a string argument"
    if len(str) == 0: return str
    newstr = ""
    for ix in xrange(len(str)):
        newstr = newstr + chr(e2aG[ord(str[ix])])
    return newstr

def Test():
    str = "The dog jumped over the lazy brown fox in 1.234567890 seconds"
    str1 = EbcDicToAscii(AsciiToEbcDic(str))
    if str != str1:
        raise "Test failed"

if __name__ == '__main__':
    Test()
```

它怎样工作

这个脚本能作为由生命支持系统围绕的两个大的查找表格来观察。当数值 `str` 被传递到脚本的两个函数之一时，脚本执行错误检验来确认数据是字符串类型，使用下面代码：

```
if type(str) != type(" "):
    raise "Bad data", "Expected a string argument"
```

在单个行中的转换的工作用这种类型的每个函数来完成：

```
newstr = newstr + chr(e2aG[ord(str[ix])])
```

注意这个脚本只使用 **Python** 的内置语句。在生成这个脚本期间没有滥用模块。

匿名者提供

你可以用两种方法从命令行得到输入：`input()`和 `raw_input()`。如果你正在寻找数字，它们之间的不同之处是很重要的。当用户键入"10"时，`raw=input("Type a number:")`将产生一个整数 10。如果你使用 `raw=raw_input("Type a number:")`就会得到同样的提示，但变量 `raw` 将包含一个字符串"10"。总之，使用 `raw_input()`比较好，所以如果你正在检索一个数字，用另一个语句转换它，例如 `int()`。

10.5 lc.py

不像本章前面的脚本那样从键盘接受输入，这个脚本从文件中取得输入。`lc.py` 在输入文件中计算行数并打印结果。

```
lc.py
```

```
...
```

计算输入文件中的行数。可以将输入文件假设为文本，但不必一定如此。

```
...
```

```
import sys, glob
```

```
def count_lines(files):
```

```
    retval = 0
```

```
    for file in sys.argv[1:]:
```

```
        if not IsFile(file): continue
```

```
        try:
```

```
            fp = open(file, "rb")
```

```
            lines = fp.readlines()
```

```
            fp.close()
```

```
    return retval
```

```

        print "%-8d %s" % (len(lines), file)
    except:
        sys.stderr.write("Couldn't read \"%s\"\n" % file)
        retval = 1
    return retval

def IsFile(file_name):
    '''如果 file_name 是一个文件，则返回 1；否则返回 0。'''
    import os
    try:
        s = os.stat(file_name)
        return ((0100000 & s[0]) == 0100000)
    except:
        return 0

if __name__ == "__main__":
    if len(sys.argv) < 2:
        print "Usage: lc file1 [file2...]"
        sys.exit(1)
    sys.exit(count_lines(sys.argv[1:]))

```

它怎样工作

这个脚本是文件打开和关闭以及 `readline()` 使用的简洁说明。文件可以用下面的行来打开：

```
fp = open(file, "rb")
```

这样会用读权限打开文件（“r”），并允许在一些平台上使用二进制文件（“b”）。下一行：

```
lines = fp.readlines()
```

从开头到结尾读取行，返回它们的列表。然后用下面的行关闭文件：

```
fp.close()
```

行的结果数计算出来并打印在一行中：

```
print "%-8d %s" % (len(lines), file)
```

匿名者提供

10.6 sample.py

这个模块能获得一个随机抽样调查数。它有丰富的注解，可能是任何程序员能为后面的

人做的最好的事情之一。

"""

本模块包含采用/不用置换和随机洗牌抽样的函数。算法来自 Knuth, 第 2 卷, 3.4.2 小节。

```
sample_wr(population_size, sample_size)
```

用由从 1 到 population_size 间的一组整数的置换进行抽样。返回选择的 sample_size 整数列表。抽样分布是二项分布。

```
sample_wor(population_size, sample_size)
```

不用由从 1 到 population_size 间的一组整数的置换进行抽样。返回选择的 sample_size 整数列表。抽样分布是超几何分布。

*

```
shuffle(sample_size)
```

返回整数 1 到 sample_size 间的随机排列。

```
deal(deck_size, num_hands, num_per_hand)
```

返回从整数 1 到 deck_size 间发牌手数（整数列表）的字典。手数用 1, 2, ..., num_hands 标记。任何剩余的“牌”都进入由 0 标记的列表。

```
deal_deck(num_hands, num_per_hand)
```

返回一发牌字典。使用了 deal() 函数，但是程序也将整数映射成了包含牌标识的字符串。例如，1 -> 2S, 2 -> 3S,

...

```
import whrandom
```

注意：创建 generator 全局变量十分重要。如果你将它放到了一个函数中，并立即调用该函数，
generator 将被使用时钟初始化。如果调用足够快，你将看到明显的非随机行为。

```
def sample_wor(population_size, sample_size):
    assert type(population_size) == type(0) and \
           type(sample_size) == type(0)    and \
```

```

        population_size > 0          and \
        sample_size > 0              and \
        sample_size <= population_size
global rand_num_generatorG
m = 0 # 至此所选记录的数量
t = 1 # 如果 frac 是正确的, 则可选择整数
list = []
while m < sample_size:
    unif_rand = rand_num_generatorG.random()
    frac = 1.0 * (sample_size - m) / (population_size - (t-1))
    if unif_rand < frac:
        list.append(t)
        m = m + 1
    t = t + 1
return list

def sample_wr(population_size, sample_size):
    assert type(population_size) == type(0) and \
           type(sample_size) == type(0)      and \
           population_size > 0                and \
           sample_size > 0
    global rand_num_generatorG
    list = []
    for ix in xrange(sample_size):
        list.append(rand_num_generatorG.randint(1, population_size))
    return list

def shuffle(sample_size):
    '''Moses 和 Oakford 算法 参阅 Knuth, 第 2 卷, 3.4.2 小节,
    返回从 1 到 sample_size 整数的随机排列。'''
    assert type(sample_size) == type(0) and sample_size > 0
    global rand_num_generatorG
    list = range(1, sample_size + 1)
    for ix in xrange(sample_size - 1, 0, -1):
        rand_int = rand_num_generatorG.randint(0, ix)

```

```

    if rand_int == ix:
        continue
    tmp = list[ix]
    list[ix] = list[rand_int]
    list[rand_int] = tmp
return list

```

```
def shuffle_set(set):
```

```
    '''返回一个已随机洗牌的序列设置的副本。用列表和元组进行。'''
```

```

    if type(set) == type({}):
        shuffled_set = []
    elif type(set) == type(()):
        shuffled_set = ()
    else:
        raise "Bad data", "must be list or tuple"
    numlist = shuffle(len(set))
    for jx in numlist:
        ix = jx - 1
        shuffled_set.append(set[ix])
    return shuffled_set

```

```
def deal(deck_size, num_hands, num_per_hand):
```

```

    ''' 返回从 1 到 deck_size 整数的发牌字典。每个字典元素（由 1, 2, ...,
    num_hands 索引）都是一列 num_per_hand 整数。由 0 索引的元素是剩余的整数，它们没有
    在手中选定。'''

```

```

    assert deck_size > 0 and \
        deck_size >= num_hands * num_per_hand and \
        num_hands > 0 and \
        type(num_hands) == type(0) and \
        num_per_hand > 0 and \
        type(num_per_hand) == type(0)
    dict = {}
    # 生成将使用的整数
    sample = shuffle_set(range(1, deck_size + 1))
    for ix in range(num_hands): # 划入字典。

```



```

        start = ix * num_per_hand
        stop = start + num_per_hand
        dict[ix+1] = sample[start : stop]
    dict[0] = sample[stop:]
    return dict

def deal_deck(num_hands, num_per_hand):
    '''返回一发牌字典。使用了 deal() 函数，但是程序也将整数映射成了包含牌标识的字符串。
    例如，1 -> 2S, 2 -> 3S, ..., 13 -> AS, 14 -> 2C 等。
    ...

    cards = deal(52, num_hands, num_per_hand)
    # 现在浏览一遍并将标识符写在牌上（并将它们从整数类型改为字符串类型）
    suit_name = "SCHD"          # 黑桃、梅花、红心、方块
    card_name = "A234567890JQK" # 注意 10 将需要特别处理
    for hand in cards.keys():
        new_hand = []
        for card_num in cards[hand]:
            # 我们从 card_num 中减 1 是因为卡的命名号是从 1~52 的
            suit_index, card_index = divmod(card_num-1, 13)
            #print "suit_index =", suit_index, " card_index =", card_index, "
card_num =", card_num
            suit = suit_name[suit_index]
            card = ("1" * (card_index -- 9)) + card_name[card_index]
            new_hand.append(card + suit)
        cards[hand] = new_hand
    return cards

```

它怎样工作

如同处理大部分模块或脚本一样的概率，这个脚本使用随机数产生器，`whrandom` 模块。尤其是以下行：

```
rand_num_generatorG = whrandom.whrandom()
```

在 `whrandom` 的 `whrandom` 类的外部创建了一个新变量 `rand_num_generatorG`。然后，`rand_num_generatorG` 被明确地声明为全局变量。通常不这样处理变量，但在这它似乎是有必要的。

这个模块的另一个不一般的特征是 `assert` 语句的使用。这条语句通常用于调试。`assert`


```
    else:
        lines[line_num] = line

# 现在将行分为单词，并建立一个单词列表
dict = {}
for line_num in xrange(len(lines)):
    if lines[line_num] == "": continue
    words = re.split(" **", lines[line_num])
    for word in words:
        if word == "": continue
        if not dict.has_key(word):
            dict[word] = []
        line_num_1_based = line_num + 1
        if line_num_1_based not in dict[word]:
            dict[word].append(line_num_1_based)
return dict

if __name__ == '__main__':
    import sys
    if len(sys.argv) != 2:
        print "Usage: xref file"
        sys.exit(1)
    words = Xref(sys.argv[1], 1)
    list = []
    # 查找最长的单词
    maxlen = 0
    for key in words.keys():
        if len(key) > maxlen:
            maxlen = len(key)
    # 现在打印输出的结果
    template = "%%-%ds: " % maxlen
    for key in words.keys():
        str = template % key
        for line_num in words[key]:
            s = "%d," % line_num
```

```
        str = str + s
    str = str[:len(str)-1] # 删除最后的逗号
    list.append(str)
list.sort()
for s in list:
    print s
```

它怎样工作

函数 `Xref` 打开一个文本文件，把文件分成由空白和标点隔离的元素，并编译结果的字典。这个模块通过使用模块 `re`（正则表达式）完成指定任务。模块 `re` 交给你操纵正则表达式的命令群。

操作从某些设置开始，输入 `re`、输入 `string` 并声明一些全局变量。然后，你会拥有一些代码行：

```
try:
    fp = open(filename, 'r')
    lines = fp.readlines()
    fp.close()
except:
    raise "Couldn't read input file \"%s\" % filename
```

这个序列使用 Python 的能力在监控的错误收集环境中运行代码行。语句 `try` 之下的行也许会也许不会正确工作，但是如果在操作期间出现了异常（像它只是假使文件不能打开），那么到 `except` 语句的执行而 `raise` 语句执行。`raise` 语句暗中导致了一种异常。

在文件打开并读取之后，所有的标点会剥离出来并用空格代替。用 `re` 模块完成这项操作是相当聪明的。`lengthy` 语句

```
punctuation_reG=re.compile("/ | \\\"|' | \\")
```

创建 `re.compile()` 的一个新对象，并用标点的每种可能的格式装载这个新对象。这个对象随后用于模式匹配。

最后，这些行被分成单词，并且这些单词被集成字典。

匿名者提供

在 Python 中检查字符串是很容易的。例如，要搜索一系列字符串，你可以这样做：

```
target="sheep"
list_of_stuff=["Victoria", "sheep", "Scotsmen"]
if target in list_of_stuff:
    print "Got it"
```

10.8 piglatin.py

记住 pig Latin, 这个有时我们生活中非常熟悉的不成对的英语回文词, 如果我们已经拥有了这个脚本, 我们只是能命令 Python 为我们做翻译。这个脚本可以在两个方面进行, 从英语到 pig Latin 并复原。

```
# piglatin.py          8/16/96  JJS
#
# piglatin(s) - 返回文本字符串的 piglatin
# depiglatin(s) - 返回 piglatin 字符串的文本
#
# 警告: 该代码输出到外国可能会构成对国家安全的侵害。
#-----

from string import splitfields, uppercase, lowercase, upper, lower

def append(w,suffix):
    if not w: return suffix
    elif w[-1] in uppercase: return w + upper(suffix)
    else: return w + lower(suffix)

def piglatin(s):
    out = ''
    for word in splitfields(s, ' '):
        if word:
            # 检查标点符号
            p = 0
            while word[p-1] not in uppercase+lowercase:
                p = p-1
            if p:
                punc = word[p:]
                word = word[:p]
            else: punc = ''

            # 以及前标点 (如圆括号)
            p = 0
```

```

        while word[p] not in uppercase+lowercase:
            p = p+1
        prepunc = word[:p]
        word = word[p:]

        # 注意大小写
        if word[0] in uppercase: caps = 1
        else: caps = 0

        # 将第一个元音前的字母删除，以生成后缀
        p = 0
        while p < len(word) and word[p] not in "aoeuiyAOEUIY":
            p = p+1

        if not p:
            word = append( word, "yay" )
        else:
            word = append( word[p:], word[:p]+"ay")

        # 重新恰当地大写
        if caps: word = upper(word[0]) + word[1:]

        #存储任何标点
        word = prepunc + word + punc

    out = out + ' ' + word
    return out[1:]

def depiglatin(s):
    out = ''
    for word in splitfields(s, ' '):
        if word:
            # 检查标点符号
            p = 0
            while word[p-1] not in uppercase+lowercase:

```

```
        p = p-1
    if p:
        punc = word[p:0]
        word = word[:p]
    else: punc = ''

    #以及前标点（如圆括号）
    p = 0
    while word[p] not in uppercase+lowercase:
        p = p+1
    prepunc = word[:p]
    word = word[p:]

    # 注意大小写
    if word[0] in uppercase: caps = 1
    else: caps = 0

    # 查找后缀
    if lower(word[-3:]) == "yay":
        word = word[:-3]

    else:
        if lower(word[-4:1]) in ['tray', 'stay', 'shay', 'play',
'quay', 'thay', 'whay']:
            suflen=4
        else: suflen=3
        if word[1] in lowercase:
            word=lower(word[0])+word[1:]
        word=word[-suflen:-2]+word[word[:]-suflen]
        if caps:
            word=upper(word[0])+word[1:]

    #存储任何标点
    word=prepunc+word+punc

    out=out+' '+word

return out[1:]
```

```

if __name__ == "__main__":
    print "PigLatin demo (enter 'quit' to quit)."
```

s - "

```

    while lower(s) != 'quit':
        s = raw_input(">")
        print "-->", piglatin(s)
        print "<--", depiglatin(piglatin(s))
    print piglatin('Thank you, have a nice day!')
```

它怎样工作

好的，这样看来它也许不是非常高技术的产品，它似乎是来自顶端的警告，但它是一个玩笑。信不信由你，许多 Web 网站都专心于 pig Latin。它们其中之一是 www.achiever.com/freemapg/cryptology/pig.html。它有转换规则。pig Latin 甚至有自己的方言。举例来说，带有连接符的方言：foo-yay 与 fooyay 刚好相反。

piglatin.py 有两个函数。一个从英文转换到 pig Latin；另一个从 pig Latin 转换到英文。这两个函数开始非常相似，并且第一项工作是向上检测标点。这个脚本用下面这些行做这项工作非常简单：

```

# 检查标点符号

p = 0
while word[p-1] not in uppercase+lowercase:
    p = p-1
if p:
    punc = word[p:]
    word = word[:p]
else: punc = ''
```

代码检测变量 word，在变量 word 中有部分输入序列，看看在字符串模块的大写字母和小写字母的列表中是否有东西。别的东西大概是标点。这和相对途径扫描已知的标点符号相反。变量 p 作为计数器使用。如果 p 在扫描操作完成后仍非零，那么就有一个标点符号写进变量 punc 中。变量 word 给出了一切别的东西。如果没有标点符号，punc 为空。

注意变量 word 中没有字符串是很重要的；它有一列字符串。这种技巧可以用下面的行来处理：

```
for word in splitfields(s, ' '):
```

这个代码输入文件 s，一次一个拔出单词并把它们放进变量 word 中。所以，输入字符串中的每个单词都分别处理。

这个代码也有一个检查“precapitalization”的例行程序，像插入语一样。

这里有两个函数分叉。对于不同的事情，它们都搜索各自的输入。piglatin(s)通过排列元音并寻找非元音来寻找辅音。

```
while p < len(word) and word[p] not in "aoeuiyAOEUIY": p=p+1
if not p:
    word.append(word, "yay")
else:
    word=append(word[p:], word[:p]+"ay")
```

在那之后，任何大写字母都会复原并且标点会被替换。depiglatin(s)的任务不同，它必须检测结尾的特定集合。然后，它必须切开 pig Latin 结尾，重构单词并复原英文结尾。

```
#查找后缀
if lower(word[-3:]) == "yay":
    word=word[:-3]
else
    if lower(word[-4:1]) in ['tray', 'stay', 'shay', 'play', 'quay', 'thay',
        'whay']:
suflen=4
    else: suflen=3
```

这些代码必须确信它没有被大小写字母的混合物 tripped，所以它明确地使用字符串模块的 lower() 程序。这些行导致变量 suflen 成为三个或四个。

其余的代码去掉了后缀，使用切片表达式，并放回大写和标点。然后从输入字符串中取得另一个单词，如果得到另一个单词，开始结束。

乘法运算符 (*) 在字符串和数字上的工作不同，但它对两者都有影响。对于数字来说，它将数字相乘。对于字符串，它复制字符串调用的次数。很遗憾，乘法运算符不能做的一件事是乘以两个字符串。

例如：

```
>>> 24.0 * 2
48.0
>>> "spam " * 2
'spam spam '
>>> "spam " * "eggs"
Traceback (innermost last):
  file "<stdin>", line 1, in ?
TypeError: can't multiply \
sequence with non-int
```

由 Joe Strout 提供，他是 California 州 La Jola 的 Salk Institute 的科学软件开发者。

第 11 章 系统操作和编程

本章要点:

- spinner.py
- tabfix.py
- python2c.py
- otp.py
- space.py
- tree.py

Python 能做一些涉及系统和编程的有兴趣事情, 例如遍历目录树或把 Python 代码翻译成 C 语言。

本章有涉及系统本身或编程方面的多种脚本。

11.1 spinner.py

...

设计一个文本微调控制器。打印到 stdout 中。

...

```
import sys
```

```
class Spinner:
```

```
    def __init__(self, type=0):
```

```
        if type == 0:
```

```
            self.char = ['.', 'o', 'O', 'o']
```

```
        else:
```

```
            self.char = ['|', '/', '-', '\\', '-']
```

```
        self.len = len(self.char)
```

```
        self.curr = 0
```

```
    def Print(self):
```

```
self.curr = (self.curr + 1) % self.len
str = self.char[self.curr]
sys.stdout.write("\b \b%s" % str)

def Done(self):
    sys.stdout.write("\b \b")

if __name__ == "__main__":
    s = Spinner()
    # 这项工作在我的老而慢的计算机上还可以，但是你必须为你的计算机而调整它。
    for jx in xrange(100):
        for ix in xrange(40000):
            pass
        s.Print()
    s.Done()
    print
```

它怎样工作

这个脚本能作为主群组或模块使用。它会在屏幕上放置一个微调控制项，让用户知道系统不紧张而是正在计算机机房中努力工作。

`spinner.py` 实际上是成为一体的两个微调控制项。在 `_init_` 的定义中，你会找到两个具有文本的列表。一个是“基于零”的微调控制项，另一个是“基于斜杠”的微调控制项。如果变量 `type` 为零，那么你会看到基于零的微调控制项。如果变量是其他东西，你会得到斜杠微调控制项。

检验下面的行：

```
sys.stdout.write("\b \b%s" % str)
def Done(self):
    sys.stdout.write("\b \b")
```

“`\b`”在做什么？它是一个换码字符、回退键。微调控制在通过屏幕生成一系列动画如“`oOo.oOo`”时做得不好，因为你每次都需要后退光标。`\b` 可以做这项工作。`%s` 是 `str` 内容中的占位符。

11.2 tabfix.py

提供者说他编写这个脚本是为了解决交叉平台的问题。由于许多年前我们热爱的 VT100

终端设备的局限性，UNIX 系统通常只能处理 8 个空格标签的文本编辑器。在其他平台上的现代文本编辑器通常有四个空格标签宽度。这种差异使共享 Python 脚本有点困难，因为 Python 毫无疑问是基于缩排的。这个脚本接受文件名、改变标签并关闭文件。

```
# 由 Joe Strout 编写的 tabfix.py
#
# 将 4 个空格标签文件转换成由 8 空格标签和 4 空格字符串组成的文件。

import sys
import string
import regex

TAB = '\t'
endspace = regex.compile(' * $')

def fixTabs(str):
    # 首先，用空格替换所有的标签
    str = string.expandtabs(str,4) # 假定 4 个空格标签
    # 现在在可能的地方将空格转换为标签
    tabsize = 8
    col = (len(str)/tabsize)*tabsize
    while col >= 0:
        # 在下一个标签停止前至少要检查空格尾端
        # 如果发现，用标签代替
        pos = col + endspace.search( str[col:col+tabsize] )
        if pos >= col:
            str = str[:pos] + TAB + str[col+tabsize:]
            col = col - tabsize
    return str

def processFile(filename):
    try: file = open( filename, 'r' ) # 打开文件
    except:
        print "Error opening",filename
        return
    lines = file.readlines() # 读取文件
    for i in range(0,len(lines)): # 处理文件
        lines[i] = fixTabs(lines[i])
```

```
file = open( filename, 'w' ) # 再次打开文件
file.writelines(lines) # 写入文件
print len(lines), "lines processed in",filename
# 结束 processFile()函数

def run(filename=''):
    if filename=='':
        filename = raw_input('Enter name of a textfile to read: ')
    processFile(filename)

# 结束 run()函数
# 用于拖拉或 execfile()执行的立即模式命令

if __name__ == '__main__':
    if len(sys.argv) == 2:
        processFile(sys.argv[1]) # 接受命令行文件名
    else:
        run()
    print raw_input("press Return")
else:
    print "Module tabfix imported."
    print "To run, type: tabfix.run()"
    print "To reload after changes to the source,type: reload(tabfix)"
# 结束 tabfix.py
```

它怎样工作

参见附带的 CD 上的自述文件中有关使用脚本运行的提示。

`tabfix.py` 有两种模式。如果你作为主程序来运行它，从命令行给它发送参数，它就开始用 `processFile()` 进行处理。你也能通过输入它并用 `tabfix.run()` 启动它，从而像实时的交互方式一样开始 `tabfix.py`。

如果脚本从具有参数的命令行中运行，它用 `processFile()` 开始处理，打开文件、读取文件的行、依次处理每一行、并将文件写回，然后关闭文件。

`fixTabs()` 是转换发生的地方。它使用字符串模块的 `expandtabs()` 函数来代替四个空格标签的间隔。下一行用一个标签代替 8 个空格。

由 Joe Strout 提供，他是 CA. La Jolla 的 Salk Institute 的科学软件开发者。

11.3 python2c.py

这个脚本阐述了有关文件转换和过滤的基础。它实际上是把有限量的 Python 代码变为相同的有限量的 C++ 代码。

如果你想按现状运行这个脚本，你需要一个命名为 `wmod` 的模块，它是特定编写的调试模块。这个模块没有包括在脚本中，但总之，它的调用行已经被注解了，所以不用担心按现实情况使用脚本。

```
# -----
# python2c.py joe@strout.net
#
# 这是一些将 Python 代码转化为 C++ 代码的小程序。它具有有限的范围，但是它在-)周围建立的
# 示例代码中工作得很好。
#
# 第一版本: 3/21/97 JJS
#
# 由 Dirk Heise 01-12-98 更改 (感谢 Dirk!)
# - 创建非常简单的文件接口
# - 一些用于 C++ 支持的规则
# - 添加另一个“动作”，以匹配中它可以包含一个执行的 Python 表达式，所以一个匹配可以触发
# 某些事
# 创建一个类头
# 几个字符串列表中的临时缓冲输入
# 添加 DEVELOPING 选项帮助检查规则的运行
# -----

import regex
import string
import wmod

# dirk heise, 是用于调试输出，而不是用于标准的模块
# 应用函数 DEBUGWRITE(str)，很好

DEVELOPING = 1

# 设置该项为 0，以便简化翻译
```

```

# 如果想要在生成的 C++ 代码中具有附加的注释, 将该项设为 1, C++ 代码可以帮助查找 PYTHON2C
# 做了些什么
# dirk heise:

# ----- RULES -----

# 所有这些规则都在输入文本的单行中起作用!
# 每个规则都由 3 个字符串组成 (第三个字符串可能是 None)
# - regex
# 如果想要创建新规则: 则 # the regex 必须包含一个或多个 "\(.*\)" 样式
# - 替换规则
# 替换规则可能指的是在上面的及的具有 "^1", "^2", "^3" 等的样式, 以插入由子表示式包括的子
# 字符串。
# - None 或 Python 语句在 match. 上执行
# 当执行该语句时, 它可以将被 regex 吃掉的子描述作为 "sub[1]"、"sub[2]" 等
# 你可以使用此将来自解析文本的信息保存为字符串变量。
# 重要信息: 正在定义的行在局部字空间内有效; 在这个程序中不能直接设置全局变量。
# (i suspect it might be a bug in Python1.4)
# 我的工作建议:
# 当你定义这样的行时, 只是调用你自己定义的函数!
# 这个函数 (参看下面的 SetClassName()) 可以访问每一个全局对象。
# 规则从列表的上部向下施加!
# 你可以通过首先捕捉特殊事例来利用它, 稍后捕捉更多生成的事例 (换言之, 序列顺序可能很重要)。

trans = [
    # COMMENTS
    # 0
    ["\(.*\)# \(.*\)", "^1//^2", None],

    # STRING LITERALS
    # 1
    ["\(.*\)' \(.*\)' \(.*\)", '^1"^2"^3', None],

    # WHILE LOOPS
    # 2

```

```

["while \(.*\):\(.*\)", "while (^1)^2", None],

# FOR LOOPS
# 循环迭代可被轻易转换的整数。
# 3
["for \(.*\) in range(\(.*\),\(.*\)):\(.*)",
  'for (int ^1=^2; ^1<^3; ^1++) {^4', None],
# 试图了解按照一些顺序/列表循环整数的意义:
# (当下面的规则是最后整数的超集时, 规则顺序是很重要的):
# 4
["for \(.*\) in \(.*\):\(.*)",
  "for (int ^1i=0; ^1i<^2.Length(); ^1i++) { int ^1 = ^2[^1i]; ^3", None],
# 这里, 我假设 Python 顺序由一些 C++ 容器所代替, 并且该容器提供了一种查找它的元素数
# 量的方法 Length()。
# 当 Python 循环不需要为此而初始化计数变量时, 重复 C++ 动态的像数组的容器时要求容器
# 初始化。并且它要求确切地访问容器的内容。该规则为此而构造了一些代码。
# 尽管它不是兼容的, 它也可以在必要时间接地通知你, 它是容易忽略的事情。
# TODO: 用一些更灵活或在某个地方定义的完全的容器接口来代替 Length()。

# IF LINES
# 5
["if \(.*\):\(.*)", "if (^1)^2", None],

# ELSE LINES
# 6
["else:\(.*)", "else^1", None],

# PRINT LINES
# 7
["print \(.*)", "$", "cout << ^1 << ' ';", None],
# 8
["print \(.*)", "cout << ^1 << endl;", None],

# INPUT STATEMENTS
# 9

```



```

['\(.*\)=\(.*\)raw_input("\(.*\)"\)\(.*\)',
  'cout << "^3"; cin >> ^1;^4', None],
# 10
["\(.*\)-\(.*\)raw_input(\(.*\))\(.*\)",
  "cin >> ^1;^4", None],
# 11
['\(.*\)=\(.*\)input("\(.*\)"\)\(.*\)',
  'cout << "^3"; cin >> ^1;^4', None],
# 12
["\(.*\)=\(.*\)input(\(.*\))\(.*\)",
  "cin >> ^1;^4", None],

# C++ RULES
# dirk heise 写的更说尽的规则
# MEMBER VARIABLE PREFIXES (TREATING "SELF.")
# 这由两个规则完成，其顺序是重要的！
# 13
# ["\(.*\)self\\.\\(\\([a-z]\\|[A-Z]\\)+\\)(\\(.*)" , "^1^2(^4" , None],
# 这个捕捉"self.id("
# 第一个捕捉函数调用对象本身，并且简单地杀死".self"
# TODO 这个 regex 失败了，为什么？ 找到一个更容易的方法来捕捉字符集
["\(.*\)self\\.\\(.*)" , "^1m_^2" , None],
# 捕捉其余部分：成员变量访问
# 这条规则假定 C++ 程序员习惯调用成员变量 "m_something"（这是我的习惯）改变这条规
# 则，使之适合你自己的 C++ 命名习惯。

# CLASS DECLARATIONS
["class \(.*\):" , " , " , "SetClassName(sub[1])"],
# 将检测到的类指定为全局字符串

# FUNCTION & METHOD DECLARATIONS
# 第一捕捉方法声明：
["def \(.*\)(self):\(.*)" , "void ^c::^1(^2", None],
["def \(.*\)(self,\(.*)" , "void ^c::^1(^2", None],
# 将类名放在函数名前，吃掉参数 "self", "void" 只是猜测。

```

```

# TODO: ^c 对于类名是非常清楚的。
# 设置"classname"很好，因为它是一个扩展的清除方法，但^C 被内嵌到了 translate 函
# 数，并不应该这样。

#
# 现在捕捉正常函数声明（它们没有"self"参数）。
["def \(.*\)" , "void ^1", None],
# again, the void is a guess.
]

# ----- EXTENSIONS -----
# 这些变量和函数用于用户定义的 Python 语句
# （参阅规则的描述）

header = [] # 只为类头存储字符串列表
            # 只在查找一类定义时使用

def hprint(s):
    # 向类头文本附加字符串
    header.append(s)

classname = ""
# dirk heise
# 保持检测类名的全局变量

def SetClassName(s):
    # dirk heise
    # 建立类名，用于用户可执行的语句中
    # 我假设在定义 Python 类时调用该项函数
    # 所以创建一些可以用作标题文件模板的代码
    global classname
    classname = s
    hprint ("VERY ROUGH HEADER FILE TEMPLATE FOR CLASS "+classname)
    hprint ("copy this into its own file and refine it by hand." )
    hprint ("// "+classname+".H" )
    hprint ("// " )
    hprint ("// " )

```

```

hprint ("# ifndef _"+classname+"_H_" )
hprint ("# define _"+classname+"_H_" )
hprint ('# include "globs.h"' )
hprint ("class "+classname )
hprint (" {" )
hprint (" public:" )
hprint ("     "+classname+"();" )
hprint ("     virtual ~"+classname+"();" )
hprint (" protected:" )
hprint (" private:" )
hprint (" };" )
hprint ("# endif // _"+classname+"_H_" )
hprint ("END OF HEADER FILE TEMPLATE FOR CLASS "+classname )

```

TODO 为什么所有的 mess 都有 hprint? 这个主意可以扩展到这里: 在破坏原型前只是写入,
 # 稍后, 当取“def NAME1”时, “打印”翻译, 并且“hprinter”行作为原型 (这样该头文件
 # 将包含更多的精确的信息), 最后, “hprint”头文件的其余部分。

dirk heise: 添加参数 exe def 翻译(s,keys,values,exe):

翻译 s 行

找到关键词间的匹配

采用 C++ 注释全局类名, 返回转换过的 's' 和一个说明施加转换次数的 history 字符串# dirk

heise

changed = 1

history = ""

history 建立了一个转换次数的字符串, 稍后我们可以查看进行了多少次转换:

changed = 0

for i in range(0,len(keys)):

if keys[i].match(s) >= 0:

找到一个匹配, 进行翻译

history = history + str(i) + " "

砍用空格分隔开了 history 字符串, 以便分后容易解析这些注释 (如果有人要这样做)

s = values[i]

我们获得了一个响应——指明的调整文本中的原料

pos = string.find(s, '^')

while pos > -1:

dirk heise: 专用: ^c 代表“classname”

```

# TODO: 这是一个不系统的匆忙的改进 (参看规则部分的说明, 选择 TODO)
if s[pos+1] == 'c' :
    # 插入"classname"到字符串中
    left = s[:pos]
    right = s[pos+2:]
    k = classname
else:
    num = string.atoi(s[pos+1:pos+2])
    # s = s[:pos] + keys[i].group(num) + s[pos+2:]
    # dirk heise : 将其分开以便它更易于理解
    left = s[:pos]
    right = s[pos+2:]
    k = keys[i].group(num)
    if k==None :
        # 提出建议:
        提出 "Error in rule: missing a \\(.*\\) pattern in regex!"
s = left + k + right

# 查找另一个插入记号:
pos = string.find(s, '^')
# dirk heise: 如果给出用户语句, 则执行它
if exe[i] <> None :
    # 在执行语句前先安装"environment"字符串:
    sub = []
    sub.append("")
    k = " "
    num = 1
    while num <> 0:
        k = keys[i].group(num)
        if k==None :
            num = 0 # to quit the loop
        else:
            num = num + 1
            sub.append(k)
# sub 现在是一包含解析的子描述的字符串列表

```

```

        exec(exe[i])
    changed = 1    # 检查更多匹配

# 特殊情况：在多数语句后添加分号
pos = string.find(s+ "//", "//")
endpos = len(string.rstrip(s[:pos])) - 1
if s <> "":
    # dirk heise: 允许规则返回空字符串
    endchar = s[endpos]
    if endpos >= 3 and s[endpos-3:endpos+1] == 'else' and \
        (endpos == 3 or s[endpos-4] in " \t"):
        # 找到悬挂的关键字-不需要分号
        return (s, " //$$$ trafos applied: "+history)
    if endpos > 0 and endchar not in '{});':
        s = s[:endpos+1] + ';' + s[endpos+1:]

return (s, " //$$$ trafos applied: "+history)
# 我使用"//$$$"作为历史字符串的标记，以在后面使用
# 自动擦去这些注释

# dirk heise: 添加参数 exe:
def processLine(s,keys,values,exe):
    # 查找缩进
    global gIndents
    qtywhitechars = regex.match("[\t ]*", s)
    if qtywhitechars > -1: whitechars = s[:qtywhitechars]
    else: whitechars = ''

    if len(whitechars) > len(gIndents[-1]):
        print gIndents[-1] + "{*
        gIndents.append(whitechars)
    else:
        while gIndents and gIndents[-1] != whitechars:
            del gIndents[-1]
            if gIndents: print gIndents[-1] + "}"

```

```

# 如果不是 gIndents: 显示 "Inconsistent indentation"
# dirk heise: 继续, 不要放弃!
if not gIndents:
    print 'WARNING! Inconsistent indentation.'
    gIndents.append(whitechars)

# dirk heise: 添另的 exe, 注意 history 返回值。
s, history = translate(s[qtywhitechars:], keys, values, exe)
return gIndents[-1] + s , gIndents[-1] + " " + history

# 建立 gKeys 和 gValues
# dirk heise: 与 gExe
gKeys = map(lambda x: regex.compile(x[0]), trans)
gValues = map(lambda x: x[1], trans)
gExe = map(lambda x: x[2], trans)
gEndWhite = regex.compile('\(.*\)\\([ \\t]*\\)$')
.

gIndents = ['']

s = ""
# Dirk Heise 12.01.97: 这样注释
# 打印 "Enter python code below, 'quit' when done."
# while s != 'quit':
#     s = raw_input()
#     print processLine(s, gKeys, gValues, gExe)

# Dirk Heise 12.01.97 : 一个非常简单的文件接口
# 由 JJS 进行平台无关性修改
s = raw_input("Enter pathname of .py file:")
try:
    f = open(s)
    lines = f.readlines()
    for s in lines:
        # wmod.DEBUGWRITE("PROCESSING <"+s+">")
        cs, history = processLine(s, gKeys, gValues, gExe)

```

```

print cs
if DEVELOPING :
    # 打印应用了打印历史的转换数

# 如果有标题类，现在输出它：
for s in header:
    print s

except IOError:
    result.SetFailure("File not found!")

```

它怎样工作

参见附带的 CD 上的自述文件中有关使用脚本运行的提示。

这是本书最长的脚本之一。许多脚本是查找并转换代码。这是一个由提供者送到脚本中的代码的示例：

```

# 这是一个测试
x = -1
while x:    # 循环直到跨越 0
    x = input('\nHow much spam?')
    if x:
        print "We have:"
        for i in range(0,x):
            print 'spam',
            if i == x-2:    # (不要忘记步长)
                print "baked beans and",
    else:
        print "Enjoy!"
# 完成所有操作!

```

响应涌出下面的 C++ 代码：

```

// 这是一个测试
x = -1;
while(x) // 循环直到跨越 0
{
    cout << "\How much spam?"; cin >> x ;
}

```

```

    if (x)
    {
        cout << "We have:" << endl;
        for (i=0; i<x; i++) {
            {
                cout << "spam" << " ";
                if (I + + x-2) // (不要忘记步长!)
                {
                    cout << "baked beans and" << " ";
                }
            }
        }
    }
    else
    {
        cout << "Enjoy!" << endl;
    }
}
// 全部完成!

```

这个脚本用一束规则开始转换。然后是本质上为查找表格的摆动列表。靠近看那些所有的列表，揭示它们都在一个称为 `trans` 的巨大的列表范围中，后面跟着函数。

因为它如此沉重的注解，这些代码很难时时注意跟踪，有时甚至是很荒谬的。为了查看条目代码，你需要下降到差不多是脚本的结尾，到达这条语句：

```
s=raw_input("Enter pathname of .py file:")
```

然后，有一条 `try` 语句检测随后的几行并处理任何异常。当你输入文件名字时，它就被打开并读取。这就是模块 `wmod` 容易进入的地方。最初的脚本有注解调试行，所以你可以不管它。

原始的脚本希望一行 Python 代码，然后翻译该代码。然后可以修改该脚本以读取整个 Python 文件。文件的行传递到函数 `processLine()` 中，随之的还有 4 个参数。其中的 3 个参数已在脚本的前面建立了 `regex` 行。

Regex 是一个能把字符串编译进用于搜索和匹配的有效的小包的模块。为得到更多的详细资料，查看和你的发行版本一起出现的文档中的具体东西。但是现在，查看下面的行：

```

gKeys = map(lambda x:regex.compile(x[0]), trans)
gValues = map(lambda x:x[1], trans)
gExe = map(lambda x:x[2], trans)
gEndWhite = regex.compile("\ (.*\ (.*\)\ ([ \t]*\)\$")

```


这些行做了大量的设置。记住这列列表命名为 `trans`。这就是使用它的地方。考虑一个来自 `trans` 的一个这样的列表：

```
["\\ (.*)\\ ' \\ (.*)\\ ' \\ (.*)\\ ", '^1"^2"^3"', None]
```

这个列表中有三个项目。第一个项目是具有你在文档中能查找到的搜索参数的搜索字符串。行：

```
gKeys = map(lambda x:regex.compile(x[0]), trans)
```

依次读取 `trans` 的每个列表。然后，拷贝每个列表中的第一列元素（语句的 `x[0]` 部分）到 `gKeys`。因此 `gKeys` 是一组编译的搜索模式。

在每个 `trans` 的列表中的另外两个列表元素（实际上是规则）拷贝到它们各自的元组，分别是 `gValues` 和 `gExe`。然而，这些没有被编译，它们作为原始字符串存在。`gEndWhite` 像编译成搜索模式的简单 `regex` 一样被初始化。

现在，搜索和匹配操作正准备认真地向前进军。所有这些值都被传递到函数 `processLine()`，这个函数能识别出缩排。当然，缩排是了解 Python 代码的关键，所以 `processLine()` 是开始必不可少的地方。

在处理完这行后，关键字和数值元组被传递到函数 `translate()`。在那里，匹配在 Python 语句和 C++ 语句之间执行。关键字和数值已经能很容易地放进字典中，而不是作为元组独立存在。无论如何，这都会解决得很好。

把所有的翻译函数简化成类和作为分离的数据文件，并把它供给变量 `trans` 是很有趣的，这样也允许脚本把 Python 翻译成其他语言。

匿名者提供

私有属性只在它们被声明的类中使用。在 Python 中，给定的私有属性前端有双下划线：`__attribute`。如果你愿意也可以添加结尾单下划线，但没有人曾经那么做过。具有前端和结尾双下划线的属性被 Python 用于内部目的。要记住 Python 实际上不能禁止访问私有属性。

11.4 otp.py

需要一些 one-time pad (OTP) 序列吗？对于需要有 128 位数字长的字母数字序列的任何应用程序都是很方便的。现在，一个生存期得到两个相等的 OTP 的机会是极大的。

该模块提供了 `otp` 对象，它可以用于 `otp` 字符串。它应该适用于不需要高安全性的情况。

可以直接调用模块作为脚本，它将想要 `otp` 字符串的打印输出号和可选的 `seed` 号。例如，`python otp.py 20` 将打印输入 20 个 `otp` 字符串。

可以向设计者传递函数。该函数采用整数参数（默认为 0），并且它必须返回一个字符串。该字符串与 MD5 算法混编，并且返回混编的十六进制的表示形式。如果没有传递收集了函数的字符串，那么使用基于 whrandom 模块的整数函数。

一旦构建了 otp 对象，调用 Get() 方法以返回一个 otp 字符串。Get() 方法可以具有一个传递到 str_function() 函数的整数参数。对于默认的函数 (GenerateString())，如果 seed 为非 0，那么启动 Wichmann-Hill 传感器并从 seed 中初始化。

每次调用 Get 都将生成一个新的随机字符串，将其发送到 md5.update() 函数中，（该函数附加它到它自己内部的它已发送的所有字符串的复本上），然后获得一个新的 MD5 混编，它转化为具有 32 位字符的十六进制表示法。

该模块可以用作提供 otp 密码的安全性，这是通过代替为 whrandom 对象生成的密码的质量随机数而完成的。可以到网络外搜索 “random number” 并找到一些硬件设备以完成此操作。

```
...

import md5, whrandom

whG = whrandom.whrandom()

def GenerateString(seed = 0):
    '''Generate a string from a four byte integer. The string is the
    4 bytes of the integer, each converted to a character.
    ...
    global whG
    if seed:      # seed != 0 表示重新启动; whrandom 从时间选择
        whG.seed(seed & 0xff,
                    (seed & 0xff00) >> 8,
                    (seed & 0xff0000) >> 16)
    n = whG.randint(0, 2**30-1)
    str = ""
    str = str + chr((n & 0xFF000000) >> 24)
    str = str + chr((n & 0x00FF0000) >> 16)
    str = str + chr((n & 0x0000FF00) >> 8)
    str = str + chr((n & 0x000000FF) >> 0)
```

```
    return str

class Otp:
    def __init__(self, str_function = GenerateString, seed = 0):
        str_function(seed) # 初始化随机数生成器
        self.m = md5.new()

    def Get(self, seed=0):
        '''Return an OTP.
        ...

        self.m.update(GenerateString(seed))
        string = self.m.digest()
        str = ""
        for ix in xrange(len(string)):
            str = str + "%02X" % ord(string[ix])
        return str

if __name__ == "__main__":
    import sys
    num = 1
    seed = 0
    if len(sys.argv) < 2:
        print "Usage: otp num_times [seed]"
        sys.exit(1)
    num = int(sys.argv[1])
    if len(sys.argv) == 3:
        seed = int(sys.argv[2])
    o = Otp(seed seed)
    for ix in xrange(num):
        print o.Get()
```

它怎样工作

这个脚本的操作实际上非常简单。它需要提交给 MD5 函数的字符串，可以从 MD5 模块中方便地得到。字符串可以由用户传递进来，或者在内部用随机数发生器模块 `whrandom` 产生。如果它从内部产生，它就被转换成字符的四个字节整数。这个脚本使用逐位运算符，

这在 Python 脚本中很罕见。行：

```
str = " "
    str = str + chr((n & 0FF000000) >> 24)
    str = str + chr((n & 0FF000000) >> 16)
    str = str + chr((n & 0FF000000) >> 8)
    str = str + chr((n & 0FF000000) >> 0)
    return str
```

采用随机数，执行具有十六进制掩码的 AND 操作，并把结果移到位单元的右边状态数。语法几乎完全和 C 语言一样。结果是累积的，就像穿过所有代码行的解释程序一样。

匿名提供者

类中的类变量和实例变量之间有不同之处。在第一个类 `Horsie` 中，`num_hooves` 是一个类变量。它将被 `Horsie` 的所有实例共享：

```
class Horsie:
    num_hooves = 4
    def figure_hooves(self):
print Horsie.num_hooves
```

在第二个类 `Horsie` 中，`num_hooves` 是一个实例变量，并且只由这个类使用：

```
class Horsie:
    def __init__(self, hooves=4):
self.num_hooves = hooves
    def hoof_report(self):
print self.num_hooves
```

11.5 space.py

想要知道在许多字典中你使用了多少空间吗？这个脚本可以告诉你。

```
'''
```

该模块提供了一个函数，它构建了在特殊目录中包含字典大小的列表。

它比 C 程序大概慢一个等级，但开发只需花少量的时间：)

```
'''
```

```
import os
```

```
listG = []
```

```
def GetTotalFileSize(dummy_param, directory, list_of_files):
    '''给出一个文件列表和它们所在的字典，向全局列表 list G 添加总体大小和字典。'''
    global listG
    currrdir = os.getcwd()
    os.chdir(directory)
    total_size = 0
    if len(list_of_files) != 0:
        for file in list_of_files:
            if file == ".." or file == ".": continue
            size = os.stat(file)[6]
            total_size = total_size + size
    listG.append([total_size, directory])
    os.chdir(currrdir)

def GetSize(directory):
    '''返回表单列表[ [a, b], [c, d], ... ]，其中 a, c, ... 是字典的全部字节的编号，
    而 b, d, ... 是字典名。所表明的字典是按降序排列并且结果按字典的大小进行分类的，最
    大的字典是列表开始处。'''
    import os
    global listG
    listG = []
    os.path.walk(directory, GetTotalFileSize, "")
    listG.sort()
    listG.reverse()

def ShowBiggestDirectories(directory):
    import re.sub
    GetSize(directory)
    # 获得字节的总数
    total_size = 0
    for dir in listG:
        total_size = total_size + dir[0]
    if total_size != 0:
        print "For directory '%s':    " % directory,
        print "[total bytes = %.1f MB]" % (total_size / (1024.0*1024))
```

```

    print "Percent"
    print "of total    Directory"
    print "-----" + "-" * 50
    not_shown_count = 0
    for dir in listG:
        dir[0] = 100.0 * dir[0] / total_size
        dir[1] = re.sub("\\\\", "/", dir[1])
        if dir[0] >= 0.1:
            print "%6.1f    %s" % (dir[0], dir[1])
        else:
            not_shown_count = not_shown_count + 1
    print "    [%d directories not shown]" % not_shown_count

if __name__ == '__main__':
    import sys
    name = sys.argv[0]
    sys.argv = sys.argv[1:]
    if len(sys.argv) != 1:
        print "Usage: %s directory_to_summarize" % name
        sys.exit(1)
    ShowBiggestDirectories(sys.argv[0])

```

它怎样工作

这个脚本阐明了 `os` 模块的几个方法。许多 Python 脚本需要涉及到目录结构，并且 `os` 模块使这项工作变得相对不那么痛苦。自从写成以来，`os` 模块就可用于 Windows、Mac 和 UNIX 机器，包括 Linux。

例如，在 `space.py` 脚本中，当前目录用 `os` 的方法 `os.getcwd` 来找到。改变当前目录为用 `os.chdir`（目录）做的目录。用 `os` 和 `os.path` 模块的许多其他的方法，适用于从过程参数到文件和目录信息的所有动作。

`os/os.path` 模块不适用于每个平台，但是，如果你离开 Windows/Mac/UNIX 轴线时就会出现警告。

注意 `os/os.path` 模块和 `sys` 模块不同。`sys` 模块基本上适用于和解释程序的环境一起工作，和与像标准输入输出这样的普通系统函数连接。也要注意，在 `space.py` 中，模块 `os` 在脚本的顶端和 `GetSize()` 函数中是输入设备。这个副本并不是必需的。

匿名者提供

11.6 tree.py

有时，你需要研究特别的目录结构。可以用这个脚本来浏览树。为了作为模块使用它，输入 `tree` 并调用函数 `tree`，其函数格式为 `tree(目录、缩进、前置字符)`。

```
...
```

```
$Id: tree.py 1.2 1998/08/30 21:24:14 donp Exp $
```

该模块定义了 `Tree()` 函数。该函数将会返回一个字符串列表，它代表了传递到函数中的字典的目录树。调用句法是：

```
Tree(dir, indent, leading_char)
```

变量 `indent` 控制了每行中的每个子目录的缩进尺寸。变量 `leading_char` 设置了列表的第一位的字符的缩进，`'|'` 可能是一个很好的选择。

如果调用模块作为脚本，那么它将打印树到传递到命令行（默认为 `'.'`）中的目录的 `stdout` 中。

```
...
```

```
def visit(list, dirname, names):
    list.append(dirname)

def Tree(dir, indent=4, leading_char="|"):
    import os, string, re
    list = []
    dir_list = []
    os.path.walk(dir, visit, list)
    list.sort()
    head = re.compile("^" + dir)
    indent_str = leading_char + " " * (indent - 1)
    for directory in list:
        if directory == ".":
            continue
        y = string.replace(directory, "\\ ", "/")
```

```

    y = head.sub(" ", y)
    fields = string.split(y, "/")
    count = len(fields) - 1
    if dir == "/":
        count = count + 1
    if fields[-1]:
        str = indent_str * count + fields[-1]
        dir_list.append(str)
    return [dir] + dir_list

if __name__ == "__main__":
    import sys
    dir_to_process = "."
    if len(sys.argv) == 2:
        dir_to_process = sys.argv[1]
    leading_char = "|"
    if sys.platform == "win32":
        leading_char = "3"
    for dir in Tree(dir_to_process, leading_char=leading_char):
        print dir

```

它怎样工作

`tree.py` 实质上把它的操作分成两部分。首先，它使用模块 `os` 检测目录结构。然后它像文本字符串一样操作目录名。模块 `re` 通过执行替换函数来提供。

脚本通过下面的行从头至尾遍历目录：

```
os.path.walk(dir, visit, list)
```

从变量 `dir` 定义的根点启动。然后，脚本把在根目录下面发现的所有目录都放进变量 `list` 中，其余的是字符串操作。列表再循环之前，它看起来像如下所示：

```
[' /home/ftp', '/home/ftp/bin', '/home/ftp/etc']
```

诸如此类。每件事的初级（和冗余）`/home/ftp` 都被分离出来，并且结果列表给出了它的缩进和前置字符。

输出不是精细的格式化树。而是调用如下重要的树：

```
tree.Tree("/home", indent=1, leading_char="|")
```

它看起来有点像：

```
['home', '|ftp', '||bin', '||etc', '||lib', '||pub']
```


等等直到结束。如果你要求遍历一定深度的嵌套目录，在涌出巨大数目的目录之前 CPU 要花费相当一段时间考虑。把返回的数值变成精细的格式化树的极好的方法是：

```
import string, tree
string.join(tree.Tree("/home", indent=1, leading_char="|"), "\n")
```

匿名者提供

第 12 章 游戏程序和人工智能

本章要点:

- logic.py
- shaney.py
- therapist.py
- lotto.py
- warmer.py
- questor.py

Python 能构建一些有趣的游戏程序。虽然它不是一个真正的 AI(人工智能)语言, Python 能制造非常少见的 AI 风格的程序。尤其是 therapist.py 能愚弄我们中间比较易受骗的一些人相当一段时间。

12.1 logic.py

这个脚本是一个猜谜游戏。这个脚本会提出一个字母, 并希望猜中它是什么, 能得到关于那个字母的问题。有个内置缺陷, 作为注解行的证明。当你读取代码时看你是否能捕捉到它。通过键入简明的字符你可以询问脚本问题。从代码本身来说, 这是一个示例会话。

Sample transcript:

```
Next? curves?
1.
Good question.
Next? c
You don't have enough information yet.
How do you know it isn't c, for example?
Next? horizontals?
0.
Next? s
You don't have enough information yet.
How do you know it isn't c, for example?
```

这个脚本是典型的几个单独的开发者提供的作品的例子。这在开放源代码 (Open

Source) 团体中是普通的事情。

```
import whrandom, string

# 推理游戏

# 选自 Judith Haris、John Swets 和 Wallace Feurzeig 编写的程序
# 参考书籍: The Secret Guide to Computers, by Russ Walter, 1993 年第18版
# 由 A.M. Kuchling (amk@magnet.com) 用 Python 编写的
# 对于每个字母, 我们都需要不同的特征: (曲线、裂丝、倾斜、水平、垂直)
# 应该有一个示例字符集以便用户参阅, 否则, 会很模糊的。例如, 是 B 有水平的特征, 还是 D? 那
么 P 和 R 呢?

# 在该数中有潜在小错误, 你能捕获它吗?
# (参阅程序底部的解答)

letter_stats={'a': (0, 2, 2, 1, 0), 'b': (2, 0, 0, 3, 1),
'c': (1, 2, 0, 0, 0), 'd': (1, 0, 0, 0, 1),
'e': (0, 3, 0, 3, 1), 'f': (0, 3, 0, 2, 1),
'g': (1, 2, 0, 1, 1), 'h': (0, 4, 0, 1, 2),
'i': (0, 2, 0, 0, 1), 'j': (1, 2, 0, 0, 1),
'k': (0, 4, 2, 0, 1), 'l': (0, 2, 0, 1, 1),
'm': (0, 2, 2, 0, 2), 'n': (0, 2, 1, 0, 2),
'o': (1, 0, 0, 0, 0), 'p': (1, 1, 0, 2, 1),
'q': (1, 2, 1, 0, 0), 'r': (1, 2, 1, 0, 1),
's': (1, 2, 0, 0, 0), 't': (0, 3, 0, 1, 1),
'u': (1, 2, 0, 0, 2), 'v': (0, 2, 2, 0, 0),
'w': (0, 2, 4, 0, 0), 'x': (0, 4, 2, 0, 0),
'y': (0, 3, 2, 0, 1), 'z': (0, 2, 1, 2, 0)}

# 我们将定义变量统计的常量, 每个常量在 letter_stats 上都与元组中统计位置是相等的,
# CURVES=0 ; LOOSE_ENDS=1 ; OBLIQUES=2 ; HORIZONTALS=3 ; VERTICALS=4
# 这个字典用于将问题映射到相应的统计量。注意不同的键可再映射到相同的值。例如,
# 'obliques' 和 'diagonals' 都映射到了 OBLIQUES 常量。
# questions={'curves':CURVES, 'looseends':LOOSE_ENDS, 'obliques':OBLIQUES,
# 'diagonals':OBLIQUES, 'horizontal':HORIZONTALS, 'verticals':VERTICALS}

# 进行单一游戏

def play_once():
```

```

# 在 0~26 间选择一个随机数
choice=26*whrandom.random()
# 将选择数字转换为字母: 0-a, 1-b, 等
choice=chr(ord('a')+choice)
#choice=raw_input('我应选择什么?')
# (用于调试)
# 我们将跟踪用户还有多少种可以利用的可能性。
# 以全部字母开始。
possibilities=string.lower("ABCDEFGHIJKLMNOPQRSTUVWXYZ")
# 我们还将跟踪已访问过的问题, 当用户重复了问题时告诉他。
asked=[]

# 不停地循环, the play_once()函数将会通过单击在循环中返回的语句退出。
while (1):
    try:
        #打印可能性
        # (用于调试)
        # 获取用户输入
        query=raw_input('Next? ')
        # 将输入转换成小写
        query=string.lower(query)
        # 删除所有非字母字符
        query=filter(lambda x: x in string.lowercase, query)
        # 删除空格
        query=string.strip(query)
    except (EOFError, KeyboardInterrupt):
        # End-Of-File: 用户
        print '\nOK; give up if you like.'
        return
    if len(query)==1:
        # 查询是一个字符长, 所以猜测
        if query not in possibilities:
            print ('Wrong! That guess is inconsistent '
                  'with the information you\'ve been given.\n'
                  'I think you made that guess just to see "what I would say.")

```

```
elif len(possibilities)>1:
    print "You don't have enough information yet."
    # 临时从概率中删除用户的猜测，然后生成一个随机字母。
    temp=filter(lambda x, query=query: x!=query, possibilities)
    r=int(random.random()*len(temp))
    print "How do you know it isn't",temp[r]+'',
    print "for example?"
else:
    # 查询在可能之中，len(possibilities)==1，所以用户是正确的
print "Yes, you've done it. Good work!" ; return

elif questions.has_key(query):
    # 获得 letter_starts 元组的字段以进行比较。
    field=questions[query]
    # 确定计算机字母的答案。
    result=letter_stats[choice][field]
    original_length=len(possibilities)
    # 排除不匹配谜语字母的可能性
    # filter(func, sequence)调用有关序列中每个元素的 func()，然后返回只包含
    # func()返回真的元素的一个新的序列对象。
    # 对于字符串，每个字符就是一个元素。不用定义一形式函数，lambda 用于创建一匿名
    # 函数（一个没有名字的函数）。
    # 函数所需的各种其他值均进行默认设置，所以它们在匿名函数范围内是可以访问的。
    possibilities=filter(lambda letter, letter_stats=letter_stats,
field=field, result=result: letter_stats[letter][field]==result, possibilities)
    new_length=len(possibilities)
    if field in asked:
        print "You asked me that already."
        print "The answer is the same as before:",
    else: asked.append(field)
    # 注意已问过这个问题
    print str(result)+'.'
    if (original_length==new_length):
        print "That was a wasted question; it did not exclude any
possibilities."
    elif (new_length < original_length/2 or new_length==1):
```

```

        print "Good question."
    else:
        print "I don't understand the question."

```

#打印指令。

```

print """This is a guessing game about capital letters. You can ask various
questions about the features of the letter: curves, loose ends, obliques (or
diagonals), horizontals, verticals. To make a guess, just enter the letter of your
choice.

```

Sample transcript:

```

Next? curves?
1.
Good question.
Next? c
You don't have enough information yet.
How do you know it isn't c, for example?
Next? horizontals?
0.
Next? s
You don't have enough information yet.
How do you know it isn't c, for example?

```

"""

#进行单人游戏。

```

play_once()
raw_input("Press Return>")

```

下面寻找错误的方案

它不是 Python 解释器可以捕捉的错误，相反，它是规范错误。

'C'和'S'拥有相同的指标：1 条曲线，2 个线端，没有斜线，水平线或垂直线。如果 C 或 S 被选作计算机字母，用户永远得不到正确答案，因为他（她）无法将概率缩减为 1！为纠正此错误，必定添加另外一个统计量，例如交叉点的个峰或封闭环的个数。尽管如此，统计量还是必须的。对于'C'和'S'有所区别上面两个建议都不起作用。你能想到区分上面两个字母的属性吗？

12.1.1 它怎样工作

这个脚本依赖字母具有 5 个属性的事实：弯曲、裂丝、倾斜、水平线和垂直线。例如，字母 Z 没有曲线，有两个裂丝、一个倾斜、两个水平线而且没有垂直线。因此，你可以定义 Z 为 (0,2,1,2,0)，并且那是在查找表格 `letter_state` 中定义它的方法。

这个游戏从用户按 `return` 键开始。然后脚本产生一个随机数并把它转换成字母。这就是转换代码：

```
# 在 0~26 间选择一个随机数，包括 choice=26*whrandom.random()
# 将随机数转换为字母：： 0-a, 1-b, etc.    choice=chr(ord('a')+choice)
```

占用一会时间查看这些行是如何工作的。变量 `choice` 装入一个数值，那是比 1 小的十进制浮点数，再乘以 26。立刻，你现在有了 1 和 26 之间的一个随机数。

然后，脚本需要把这个数字转换为字母。行：

```
choice=chr(ord('a')+choice)
```

使变量 `choice` 等于一个字符，这个字符是小写字母 a 加上 `choice` 的 ASCII 等价物，无论 `choice` 是什么。那样会使 a 成为一条基准 (baseline)，导致字母的恰当的随机抽样。

好的，有关抽选过程就到此为止。现在，脚本必须对用户进行交互。行：

```
query=raw_input('Next? ')
```

提示用户输入。输入存在于变量 `query` 中，然后用下面这些行过滤：

```
# 将输入转换为小写
```

```
query=string.lower(query)
```

```
# 除去所有非字母的字符
```

```
query=filter(lambda x: x in string.lowercase, query)
```

```
# 除掉空格
```

```
query=string.strip(query)
```

如果用户输入引起一个 EOF (文件结束) 错误或者如果有一次键盘中断，这些行后的 `except` 语句就会捕捉到它。

用户能键入两个不同的有效响应。第一个是像 `curves` 这样的单词来查看字母是否有任何曲线，或者用户能键入一个 `guess` (猜测的字母)。

这个脚本可以用下面的检测变量 `query` 的内容的长度的行，来查看输入是否是一个猜测的字母：

```
if len(query) == 1:
```

如果长度为 1，输入就是一个猜测的字母，在后面的逻辑中有一个有趣的交叉。稍后，脚本清楚了剩余可能性的总数。如果可能性不只一个，假如用户猜错了，游戏就会有点取笑用户。游戏从容纳所有有效的剩余可能性的变量 `possibilities` 中删除了用户的猜测，从变量 `possibilities` 中检索随机字母，并呈现给用户一句话“你如何知道它不是...？”

既然这样，困难在于是变量 `possibilities` 怎样检验出现的猜测的字母。可以用下面这个代码完成：

```
temp=filter(lambda x, query=query: x!=query, possibilities)
```

这是一个 Python `lambda` 函数，在大多数 Python 的书籍和文档中提供的解释非常少。首先，那是因为它只有有限的用途。`lambda` 函数是只有一次的未命名的函数。但是如果你需要的是只有一次的函数，它就是很理想的。

这里，`lambda` 和 `filter` 函数一起使用。在前面的代码行中，变量 `temp` 装载变量 `possibilities` 上的过滤操作的结果。在 `filter` 函数中，冒号左边是声明，右边是表达式。注意使 `query` 等于 `query` 的奇怪的代码。因为 Python 只有两个作用域，所以作为工作区来说这是必需的：局部和全局。`lambda` 操作确实不是两者中任意一个，所以为了传进或传出数值，你必须声明一个变量以便于 `lambda` 函数能操作它。既然这样，赋予 `lambda` 函数的参数的名字和变量的名字相同。两个名字不必恒等。虽然实际上，它们通常是恒等的。

如果用户的输入是一个单词，它就被假设为信息请求并传递到 `if` 树的 `elif` 部分。这部分也有一个 `lambda` 滤波函数。至于这个缺陷，游戏不能区别小写字母 `c` 和小写字母 `s`，因为它们有相同的特征：1,2,0,0,0。但是难道 `s` 不是有两上弯曲和一个倾斜吗？既然这样，你能把 `s` 定义为 2,3,1,0,0，这样就可以和 `c` 区分了。

由 Joe Strout 提供，他是 CA 的 La Jolla 的 Salk Institute 中的科学软件开发者。

如果你运行这个脚本有一些困难，尝试一下以下的这个版本，由 Mitch Chapman 好意推荐：

```
#字母逻辑游戏
import whrandom, string
# 逻辑游戏
# 由 Judith Harris、John Swets 和 Wallace Feurzeig 编写的程序
# 参考资料：计算机秘密指南，由 Russ Walter 编著，1993 年第 18 版
# 由 A.M. Kuchling (amk@magnet.com) 用 Python 编写
# 对于每个字母都需要各种特征：
# (curves, loose ends, obliques, horizontals, verticals).
# 应该有助于用户查看的抽样字符设置，否则会有歧义。例如，B 有水平线吗？D 呢？P 和 R 怎么样
# 呢？
# 数据中隐藏着一个错误（你能找到它吗？答案参看程序的底部）
letter_stats={'a': (0, 2, 2, 1, 0), 'b': (2, 0, 0, 3, 1),
              'c': (1, 2, 0, 0, 0), 'd': (1, 0, 0, 0, 1),
              'e': (0, 3, 0, 3, 1), 'f': (0, 3, 0, 2, 1),
              'g': (1, 2, 0, 1, 1), 'h': (0, 4, 0, 1, 2),
              'i': (0, 2, 0, 0, 1), 'j': (1, 2, 0, 0, 1),
```



```
'k': (0, 4, 2, 0, 1), 'l': (0, 2, 0, 1, 1),
'm': (0, 2, 2, 0, 2), 'n': (0, 2, 1, 0, 2),
'o': (1, 0, 0, 0, 0), 'p': (1, 1, 0, 2, 1),
'q': (1, 2, 1, 0, 0), 'r': (1, 2, 1, 0, 1),
's': (1, 2, 0, 0, 0), 't': (0, 3, 0, 1, 1),
'u': (1, 2, 0, 0, 2), 'v': (0, 2, 2, 0, 0),
'w': (0, 2, 4, 0, 0), 'x': (0, 4, 2, 0, 0),
'y': (0, 3, 2, 0, 1), 'z': (0, 2, 1, 2, 0)}
```

```
# 我们将为各种统计量定义常量，每个常量作为在 letter_start 元组中统计量的位置。
# CURVES=0 ; LOOSE_ENDS=1 ; OBLIQUES=2 ; HORIZONTALS=3 ; VERTICALS=4
# 该字典用于将问题映射相应的统计量。注意不同的键可能映射相同的值。例如，'obliques'和
# 'diagonals'都映射 OBLIQUES 常量。
# questions={'curves':CURVES, 'looseends':LOOSE_ENDS, 'obliques':OBLIQUES,
# 'diagonals':OBLIQUES, 'horizontal':HORIZONTALS, 'verticals': VERTICALS}
```

```
# 进行单人游戏
```

```
def play_once():
    # 从 0~26 间选择一个随机数
    choice=26*whrandom.random()
    # 将选择的数字转换成字母： 0-a, 1-b, 等
    choice=chr(ord('a')+choice)
    #choice=raw_input("我应选择什么?") # (用于调试)
    # 我们将跟踪用户还有多少种可用的可能性。
    # 以全部字母开始。
    possibilities=string.lower("ABCDEFGHIJKLMNOPQRSTUVWXYZ")
    # 我们还将跟踪已经问过的问题，等用户重复问题时告诉他。
    asked=[]

    # 一直循环，键入 Return 键，play fun()函数退出循环。
    while (1):
        try:
            #打印可能性
            # (用于调试)
```

```
# 获取用户输入
query=raw_input('Next? ')
# 将输入转换成小写
query=string.lower(query)
# 删除所有非字母字符
query=filter(lambda x: x in string.lowercase, query)
# 删除空格
query=string.strip(query)
except (EOFError, KeyboardInterrupt):
    # End-Of-File:the user
    print '\nOK; give up if you like.'
    return
if len(query)==1:
    # 查询是一个字符长, 所以猜测。
    if query not in possibilities:
        print ("Wrong! That guess is inconsistent "
              "with the information you've been given.\n"
              "I think you made that guess just to see "
              "what I would say.")
    elif len(possibilities)>1:
        print "You don't have enough information yet."
        # 临时从可能性中删除用户的猜测, 然后生成一个随机数。
        temp=filter(lambda x, query=query: x!=query, \
                    possibilities)
        r=int(random.random()*len(temp))
        print "How do you know it isn't",temp[r]+', ',
        print "for example?"
    else:
        # 查询在可能之中, len(possibilities)==1, 所以用户正确。
        print "Yes, you've done it. Good work!"
        return
elif questions.has_key(query):
    # 获得 letter_stats 元组的字段进行比较。
    field=questions[query]
    # 确定计算机字母的答案。
```

```

result=letter_stats[choice][field]
original_length=len(possibilities)
# 排除与秘密字母不匹配的可能性。
# filter(func, sequence)调用有关序列中每个元素的 func()，然后返回一个包
# 含 func()返回真的元素的新序列对象。
# 对于字符串，每个字符就是一个元素。不用定义一形式函数，lambda 用于创建一匿名
# 函数（一个没有名字的函数）。
# 函数所需的各种其他值均进行默认设置，所以它们在匿名函数范围内是可以访问的。
possibilities=filter(lambda letter, \
letter_stats=letter_stats,
                      field=field, result=result:
                      letter_stats[lotter][field]==result,\
possibilities)
new_length=len(possibilities)
if field in asked:
    print "You asked me that already."
    print "The answer is the same as before:",
else: asked.append(field)
# 注意这个问题已问过了
print str(result)+'.'
if (original_length==new_length):
    print 'That was a wasted question; it did not \
exclude any possibilities.'
elif (new_length < original_length/2 or new_length--1):
    print "Good question."
else:
    print "I don't understand the question."

```

#打印指令

print ""这是一个猜大写字母的猜迷游戏。你可以问关于字母特征的各种问题：曲线数、开口数、斜线数（或对角线数）、水平线数、垂直线数，要想猜迷，直接输入猜中的字母即可。

Sample transcript:

Next? curves?

1.

```

    Good question.
    Next? c
    You don't have enough information yet.
    How do you know it isn't c, for example?
    Next? horizontals?
    0.
    Next? s
    You don't have enough information yet.
    How do you know it isn't c, for example?
"""
#进行单人游戏
play_once()
raw_input("Press Return>")

```

12.2 shaney.py

依照这种惯例，几年前在称为 `net.singles` 的小组中 Mark Shaney 就是一个参与者。他的回答是，按照列表成员，或者是逐渐渗透或者又是一个列表。据说，“Mark V. Shaney” 实际上是一个运行在贝尔实验室服务器上的一个程序。这是一种重新创作有些 Shaney 魔术的尝试。

下面的代码是 Joe Strout 的最初的脚本。我对它作了修改，并且我的修改直接出现在 Joe 的版本后面。

```

# Greg McFarlane 编写的 shaney.py
# Joe Strout 最初的编辑
#
# 在 WWW 上搜索 "Mark V. Shaney" 以获得更多信息。

import sys
import rand
import string

def run(filename=''):
    if filename=='':
        file = open( raw_input('Enter name of a textfile to read:'), 'r')
    else:

```

```
    file = open( filename, 'r')
text = file.read()
file.close()
words = string.split(text)
end_sentence = []
dict = {}
prev1 = ''
prev2 = ''
for word in words:
    if prev1 != '' and prev2 != '':
        key = (prev2, prev1)
        if dict.has_key(key):
            dict[key].append(word)
        else:
            dict[key] = [word]
            if prev1[-1:] == '.':
end_sentence.append(key)
        prev2 = prev1
        prev1 = word
if end_sentence == []:
    print 'Sorry, there are no sentences in the text.'
    return
key = ()
count = 10
while 1:
    if dict.has_key(key):
        word = rand.choice(dict[key])
        print word,
        key = (key[1], word)
        if key in end_sentence:
            print
            count = count - 1
            if count <= 0:
                break
    else:
```

```

        key = rand.choice(end_sentence)

# run() 函数结束
# 立即模式指令用于执行拖拉或 execfile()

if __name__ == '__main__':
    if len(sys.argv) == 2:
        run(sys.argv[1]) # accept a command-line filename
    else:
        run()

    print
    raw_input("press Return")
else:
    print "Module shaney imported."
    print "To run, type: shaney.run()"
    print "To reload after changes to the source,type:reload(shaney)"

```

shaney.py 结束

我的版本从这开始。我从 `rand` 到 `random` 改变了模块的名字。在 Python 的发行版本中不再有 `rand` 模块。

```

# Greg McFarlane 编写的 shaney.py
# Joe Strout 的一些编辑
# Tim Altom 的进一步的编写
# 在 WWW 上搜索 "Mark V. Shaney" 以获得更多的信息。

import sys
import random
import string

def run(filename=''):
    if filename=='':
        file = open( raw_input('Enter name of a textfile to read:'), 'r')
    else:
        file = open( filename, 'r')
    text = file.read()
    file.close()

```

```
words = string.split(text)
end_sentence = []
dict = {}
prev1 = ''
prev2 = ''
for word in words:
    if prev1 != '' and prev2 != '':
        key = (prev2, prev1)
        if dict.has_key(key):
            dict[key].append(word)
        else:
            dict[key] = [word]
            if prev1[-1:] == '.':
end_sentence.append(key)
        prev2 = prev1
        prev1 = word
if end_sentence == []:
    print 'Sorry, there are no sentences in the text.'
    return
key = ()
count = 10
while 1:
    if dict.has_key(key):
        word = random.choice(dict[key])
        print word,
        key = (key[1], word)
        if key in end_sentence:
            print
            count = count - 1
            if count <= 0:
                break
    else:
        key = random.choice(end_sentence)
# run()函数结束
#立即模式指令用于执行拖拉或 execfile()
```

```
if __name__ == '__main__':
    if len(sys.argv) == 2:
        run(sys.argv[1]) # accept a command-line filename
    else:
        run()
    print
    raw_input("press Return")
else:
    print "Module shaney imported."
    print "To run, type: shaney.run()"
    print "To reload after changes to the source,type:reload(shaney)"

# shaney.py 结束
```

12.2.1 它怎样工作

shaney.py 的操作是非常简明的。只有输出是奇怪的。这个脚本使用随机数发生器 `modulerandom`，来决定从打开文件中打印文本的顺序。较旧的脚本也许已经使用了模块 `random`，但现今许多脚本使用模块 `whrandom`。

用于较长文件的效果更加有趣。一行或两行的文件远没有同样多的趣事。它使我想知道你是否不能分出一个相似的 Shaney 脚本给 `dictionary` 模块（顺便说一下，不是 Python 提供的），来产生确实奇妙的和意外的结果。这个脚本遍历字典，寻找随机产生的到处走访或者代替整个短语的单词，你会得到用下面这样的常规文本开始的可爱的伪哲学：

This has nothing to do with the subject at hand, you know.

并且逐步发展为下面这样：

something is a bean with the subject at hand.

你是挑选：鲜明还是谬论？在你应答之前，核对从 Bob Dylan 到 John Lennon 任一首抒情歌曲的歌词。“I an the walrus...?”，如果感谢消息的发送人是不正常的，那么任何人都知道它只是一个 Python 的脚本吗？

由 Joe Strout 提供，他是 CA 的 La Jolla 的 Salk Institute 中的科学软件开发者。

解开包是真正节省时间的事物。如果你有一个列表或元组并且想把它数值散开进入许多变量中。你能很快地像下面这样完成这项工作：

```
a,b,c=[1,2,3]
```

变量 `a`、`b` 和 `c` 已经分别装载了 1、2 和 3。举例来说，这似乎看起来并不是非常有帮助的，直到你启动了从数据库中挑选的名字的解开包列表。

12.3 therapist.py

AI 程序员的 Holy Grail 是一个程序，当向它提出问题时，不能与真正的人区分开。这就是著名的按 Alan Turing 命名的“图灵测试”，Alan Turing 是二十世纪早期的优秀数学家和计算幻想家，他也是一个不幸的人。therapist.py 不是要传递图灵测试，而是开个玩笑。

therapist.py 是一种 Eliza 程序。Eliza 是 Joseph Weizenbaum 在 1960 年中期学习 AI 时编写的。它由这句话开始“How are you feeling today? ”，你键入应答，并且脚本从它存储的回答中做出一个回答。脚本的回答似乎更像一个无聊的治疗专家反作用于病人的唠叨一样。

正如它的作者注释的一样，therapist.py 很容易受人愚弄，但它可以供人娱乐。

```
#-----
# Joe Strout 编写的有趣的小 Eliza, Jeff Epler 作了一些更新, 最后的修改: 3/17/97
#-----

import string
import regex
import whrandom

#-----
# translate: 取出一字符串, 用相应的 dict.keys() 代替在 dict.values() 中的任何单词,
#-----
def translate(str,dict):
    words = string.split(string.lower(str))
    keys = dict.keys();
    for i in range(0,len(words)):
        if words[i] in keys:
            words[i] = dict[words[i]]
    return string.join(words)

#-----
# respond: 取出一个字符串, 一组 regexps 以及一组相应的响应列表, 找到一个匹配, 然后从相
# 应的列表中返回一随机选择的应答。
#-----

def respond(str,keys,values):
    # 在关键字中找到一匹配
```

```

    for i in range(0, len(keys)):
        if keys[i].match(str) > 0:
            # 找到一匹配 ... 用相应的值填充
            # 从可用的选项中选择随机数
            respnum = whrandom.randint(0, len(values[i])-1)
            resp = values[i][respnum]
            # 我们已得到一响应...在指示的地方反映文本
pos = string.find(resp, '%')
            while pos > -1:
                num = string.atoi(resp[pos+1:pos+2])
                resp = resp[:pos]+\
                    translate(keys[i].group(num), gReflections) + \
                    resp[pos+2:]
                pos = string.find(resp, '%')
            # 在末尾填加句点。
            if resp[-2:] == '?.': resp = resp[:-2] + '.'
            if resp[-2:] == '??': resp = resp[:-2] + '?'
            return resp

#-----
# gReflections, 翻译表, 用于将你所说的翻译成计算机应对的内容。例如, "I am" -- "you are"
#-----

gReflections =
"am" : "are",
"was" : "were",
"i" : "you",
"i'd" : "you would",
"i've" : "you have",
"i'll" : "you will",
"my" : "your",
"are" : "am",
"you've" : "I have",
"you'll" : "I will",
"your" : "my",

```

```
"yours" : "mine",
```

```
"you" : "me",
```

```
"me" : "you" }
```

```
#-----
```

```
# gPats, 主响应表。列表的每一元素都是一二元列表，第一项是 regexp，第二项是可能响应的列表，分别用%1, %2 等标出。
```

```
#-----
```

```
gPats =
```

```
["I need \(.*\)",
```

```
["Why do you need %1?",
```

```
"Would it really help you to get %1?",
```

```
"Are you sure you need %1?"]],
```

```
["Why don't you \(.*\)",
```

```
["Do you really think I don't %1?",
```

```
"Perhaps eventually I will %1.",
```

```
"Do you really want me to %1?"]],
```

```
["Why can't I \(.*)",
```

```
["Do you think you should be able to %1?",
```

```
"If you could %1, what would you do?",
```

```
"I don't know -- why can't you %1?",
```

```
"Have you really tried?"]],
```

```
["I can't \(.*)",
```

```
["How do you know you can't %1?",
```

```
"Perhaps you could %1 if you tried.",
```

```
"What would it take for you to %1?"]],
```

```
["I am \(.*)",
```

```
["Did you come to me because you are %1?",
```

```
"How long have you been %1?",
```

```
"How do you feel about being %1?"]],
```

```
["I'm \(.*\)",  
 [ "How does being %1 make you feel?",  
  "Do you enjoy being %1?",  
  "Why do you tell me you're %1?",  
  "Why do you think you're %1?" ] ],
```

```
["Are you \(.*\)",  
 [ "Why does it matter whether I am %1?",  
  "Would you prefer it if I were not %1?",  
  "Perhaps you believe I am %1.",  
  "I may be %1 -- what do you think?" ] ],
```

```
["What \(.*\)",  
 [ "Why do you ask?",  
  "How would an answer to that help you?",  
  "What do you think?" ] ],
```

```
["How \(.*\)",  
 [ "How do you suppose?",  
  "Perhaps you can answer your own question.",  
  "What is it you're really asking?" ] ],
```

```
["Because \(.*\)",  
 [ "Is that the real reason?",  
  "What other reasons come to mind?",  
  "Does that reason apply to anything else?",  
  "If %1, what else must be true?" ] ],
```

```
["\(.*\) sorry \(.*\)",  
 [ "There are many times when no apology is needed.",  
  "What feelings do you have when you apologize?" ] ],
```

```
["Hello\(.*\)",  
 [ "Hello... I'm glad you could drop by today.",
```

```
"Hi there... how are you today?",
"Hello, how are you feeling today?" ]],
```

```
["I think \(.*\)",
 [ "Do you doubt %1?",
  "Do you really think so?",
  "But you're not sure %1?" ]],
```

```
["\(.*\) friend\(.*\)",
 [ "Tell me more about your friends.",
  "When you think of a friend, what comes to mind?",
  "Why don't you tell me about a childhood friend?" ]],
```

```
["Yes",
 [ "You seem quite sure.",
  "OK, but can you elaborate a bit?" ]],
```

```
["\(.*\) computer\(.*\)",
 [ "Are you really talking about me?",
  "Does it seem strange to talk to a computer?",
  "How do computers make you feel?",
  "Do you feel threatened by computers?" ]],
```

```
["Is it \(.*\)",
 [ "Do you think it is %1?",
  "Perhaps it's %1 -- what do you think?",
  "If it were %1, what would you do?",
  "It could well be that %1." ]],
```

```
["It is \(.*\)",
 [ "You seem very certain.",
  "If I told you that it probably isn't %1, what would you feel?" ]],
```

```
["Can you \(.*\)",
 [ "What makes you think I can't %1?",
```

```
"If I could %1, then what?",
"Why do you ask if I can %1?" ]],

["Can I \(.*\)",
[ "Perhaps you don't want to %1.",
"Do you want to be able to %1?",
"If you could %1, would you?" ]],

["You are \(.*\)",
[ "Why do you think I am %1?",
"Does it please you to think that I'm %1?",
"Perhaps you would like me to be %1.",
"Perhaps you're really talking about yourself?" ]],

["You're \(.*\)",
[ "Why do you say I am %1?",
"Why do you think I am %1?",
"Are we talking about you, or me?" ]],

["I don't \(.*\)",
[ "Don't you really %1?",
"Why don't you %1?",
"Do you want to %1?" ]],

["I feel \(.*\)",
[ "Good, tell me more about these feelings.",
"Do you often feel %1?",
"When do you usually feel %1?",
"When you feel %1, what do you do?" ]],

["I have \(.*)",
[ "Why do you tell me that you've %1?",
"Have you really %1?",
"Now that you have %1, what will you do next?" ]],
```

```
["I would \(.*\)",  
 [ "Could you explain why you would %1?",  
  "Why would you %1?",  
  "Who else knows that you would %1?" ] ],  
  
["Is there \(.*\)",  
 [ "Do you think there is %1?",  
  "It's likely that there is %1.",  
  "Would you like there to be %1?" ] ],  
  
["My \(.*\)",  
 [ "I see, your %1.",  
  "Why do you say that your %1?",  
  "When your %1, how do you feel?" ] ],  
  
["You \(.*\)",  
 [ "We should be discussing you, not me.",  
  "Why do you say that about me?",  
  "Why do you care whether I %1?" ] ],  
  
["Why \(.*\)",  
 [ "Why don't you tell me the reason why %1?",  
  "Why do you think %1?" ] ],  
  
["I want \(.*\)",  
 [ "What would it mean to you if you got %1?",  
  "Why do you want %1?",  
  "What would you do if you got %1?",  
  "If you got %1, then what would you do?" ] ],  
  
["\(.*\) mother\(.*\)",  
 [ "Tell me more about your mother.",  
  "What was your relationship with your mother like?",  
  "How do you feel about your mother?",  
  "How does this relate to your feelings today?",
```

```
"Good family relations are important."]],
```

```
["\(.*\) father\(.*\)",
```

```
[ "Tell me more about your father.",
```

```
"How did your father make you feel?",
```

```
"How do you feel about your father?",
```

```
"Does your relationship with your father relate to your feelings today?",
```

```
"Do you have trouble showing affection with your family?"]],
```

```
["\(.*\) child\(.*\)",
```

```
[ "Did you have close friends as a child?",
```

```
"What is your favorite childhood memory?",
```

```
"Do you remember any dreams or nightmares from childhood?",
```

```
"Did the other children sometimes tease you?",
```

```
"How do you think your childhood experiences relate to your feelings today?"]],
```

```
["\(.*\)\?",
```

```
[ "Why do you ask that?",
```

```
"Please consider whether you can answer your own question.",
```

```
"Perhaps the answer lies within yourself?",
```

```
"Why don't you tell me?"]],
```

```
["quit",
```

```
[ "Thank you for talking with me.",
```

```
"Good-bye.",
```

```
"Thank you, that will be $150. Have a good day!"]],
```

```
["\(.*\)",
```

```
[ "Please tell me more.",
```

```
"Let's change focus a bit... Tell me about your family.",
```

```
"Can you elaborate on that?",
```

```
"Why do you say that %1?",
```

```
"I see.",
```

```
"Very interesting.",
```

```
"%1.",
```



```

"I see. And what does that tell you?",
"How does that make you feel?",
"How do you fee! when you say that?"] ]

#-----
# 主程序
#-----

gKeys = map(lambda x:regex.compile(x[0]),gPats)
gValues = map(lambda x:x[1],gPats)

print "Therapist\n-----"
print "Talk to the program by typing in plain English, using normal upper-"
print 'and lower-case letters and punctuation. Enter "quit" when done.'
print '='*72
print "Hello. How are you feeling today?"
s = ""
while s != "quit":
    try: s = raw_input(">")
    except EOFError:
        s = "quit"
        print s
    while s[-1] in "!.,: s = s[:-1]
    print respond(s,gKeys,gValues)

```

它是怎样工作的

大部分脚本都是查找。代码比较用户的输入语句和输入中的某些单词，从查找表格中存储的2个到4个可能的应答中返回随机选择的回答。

这个程序工作的基础是大量会话式英语，而不是正式性英语。当我们构造句子时，我们倾向于依赖确定的开头（opening）单词和短语：I am, I'm, My 等。那些应答只有狭窄的可能回答范围，并且我们期待它们和适当的可预言的措词起反应：“Yes, I can see that you are not happy today”和“I'm sorry. Can I help?”等。我们对不能预言的回答很吃惊，例如“Shoes. You need to buy shoes”。但是我们对令人厌烦的、索然无味的并且甚至是毫无帮助的回答并不感到奇怪。我们能处理那些情况。

这个程序挑选特别的单词和短语作为已经键入的语句的线索。例如，开始时猜测某人正

在询问一个问题。这认为这是 `belief`（信任）语句的序言。查找表格 `gPats` 容纳了这个交换的关键字。第一个元素是来自用户键入的句子的重要的单词或短语，作为关键字使用。当和任何关键字匹配时，程序从可能的二个到四个答案中挑选，可能添加一个或两个变量。

脚本通过把 `gPats` 列表中的优先元素的 `regex` 编译模式放在一起开始。有脚本用于定位匹配的关键字单词和短语。至于 `gPats` 元素的其余部分，它们和另一个变量相映射。实际上，现在脚本把 `gPats` 中的每个列表分成两个分支：关键字和可能的回答。代码如下：

```
gKeys = map(lambda x:regex.compile(x[0]),gPats)
```

```
gValues = map(lambda x:x[1],gPats)
```

开玩笑地，在主程序中添加几个打印行，如下所示：

```
gKey = map(lambda x:regex.compile(x[0]),gPats)
```

```
gValues = map(lambda x:x[1],gPats)
```

```
print "gKeys:", gKeys
```

```
print "gValues:", gValues
```

这些行没有保留，它们只是从事教育上的目的。当运行这个脚本时，你会看出虽然 `gValues` 是一个典型的列表（即使真正长的列表！），`gKeys` 是一列 `regex` 对象。参见 `Python` 文档以得到更多关于 `regex` 的信息。目前只需记住 `regex` 大量加速了处理。

`regex` 是很有价值的，然而你应该记住如今在支持模块 `re` 的应用中放弃使用 `regex`。这两者的关键是常规表达式的使用。现在，主程序捆绑了用户的输入字符串，`gKeys` 和 `gValues`，并把它发送到 `respond()` 函数。当它们到达时，就各自成为 `str`、关键字和数值。`respond()` 检查匹配。当它发现一个匹配时，这个函数产生一个随机数，并使用它来选择脚本的回答。然后，回答被集合并打印出来。

要注意的一行是：

```
print '='*72
```

显然，那是整数乘以字符的结果。并且这就是所发生的事。这行代码在屏幕上打印等号 72 次。最后，你可以考虑通过稍微改写主代码来更新脚本。你可以让脚本决定它作为模块还是做为主程序来调用：

```
if __name__ == "__name__":
```

```
gKeys = map(lambda x:regex.compile(x[0]),gPats)
```

```
gValues = map(lambda x:x[1],gPats)
```

```
print "Therapist\n-----"
```

```
print "Talk to the program by typing in plain English, using normal upper-"
```

```
print "and lower-case letters and punctuation. Enter "quit" when done."
```

```
print '='*72
```

```
print "Hello. How are you feeling today?"
```

```

s = " "
while s != "quit":
    try: s = raw_input(">")
except EOFError:
    s = "quit"
    print s
while s[-1] in "!.": s = s[:-1]
print respond(s,gKeys,gValues)

```

由 Joe Strout 提供，他是 CA 的 La Jolla 的 Salk Institute 中的科学软件开发者。

12.4 lotto.py

让脚本挑选你的彩票数字，6 个为一组。它阐明了许多字符串函数，并实际上具有挑选数字的用途。

```

# manny juan 收集的排号码牌游戏 (juanm@wellsfargo.com or manny@bdt.com)
from whrandom import randint
def pick_lotto():
    maxm=53
    maxj=6
    m=maxm
    #创建从 0 到 m 的所有数
    r=range(m+1)
    #从一空结果开始
    v=[]
    for j in range(maxj):
        # 从结果中获取第 i 个
        i=randint(1,m)
        n=r[i]
        #从结果中删除
        r[i:i+1]=[]
        m=m-1
        # 追加到结果上
        v.append(n)
    return v
def run():

```

```

done=0
while not done:
    try: x=raw_input('\npress Enter for Lotto picks (Q to quit). ')
    except EOFError:
        x = 'q'
    if x and (x[0] == 'q' or x[0] == 'Q'):
        done=1
        print 'done'
    else:
        print pick_lotto()
# 立即模式指令，用于执行拖拉或 execfile()
if __name__ == '__main__':
    run()
else:
    print "Module lotto imported."
    print "To run, type: lotto.run()"
    print "To reload after changes to the source, type: reload(lotto)"
# lotto.pys 结束

```

它怎样工作

用 `pick_lotto()` 完成的实际数字选择。这里，脚本创建了从 1 到 53 的一系列数字，把它们放进一个列表，生成一个随机整数，并作为索引使用随机整数从范围列表中挑选这些数字中的一个数字。脚本完成 6 次这项工作，然后返回结果。

这同样的一般概念可以延伸，例如，产生适用于随意创建的口令的字母和数字字符串。用每个按键你能得到另一个随机口令。如果你给数千个职员发行口令，可以在数秒内结束这项工作。

由 Joe Strout 提供，他是 CA 的 La Jolla 的 Salk Institute 中的科学软件开发者。

12.5 warmer.py

`warmer.py` 是旧式 `warmer/colder` 游戏的 Python 实现。看看它要花费多少次猜测随机产生的数字。

```

# warmer.py 3/11/96 by Joe Strout
import rand # 处理随机数函数
def run():

```

```
# 在 1-100 内选择一数
mynum = rand.choice( range(100) ) + 1
yourguess = 200      # 用户猜测什么
lastdist = 0        # 我的数的最后一个实例
tries = 0           # 所尝试过的数
print "I'm thinking of a number from 1 to 100."
# 主循环: 重复直至用户获得正确答案
while yourguess != mynum:
    tries = tries + 1
    yourguess = input("Your guess? ")
    if (yourguess != mynum):
        # 找到你现在的差距
        newdist = abs(yourguess - mynum)
        # 打印和上次相比更暖/更冷的结果
        if (lastdist == 0):
            print "Guess again..."
        elif (newdist > lastdist):
            print "You're getting colder."
        else:
            print "You're getting warmer."
        lastdist = newdist
    # if 表达式结束
    # 一直重复直至用户得到正确结果
    print "Good job! That took", tries, "tries."
# 立即模式指令, 用于执行拖拉或 execfile()
if __name__ == '__main__':
    run()
    print
    raw_input("press Return")
else:
    print "Module warmer imported."
    print "To run, type: warmer.run()"
    print "To reload after changes to the source, type: reload(warmer)"
# warmer.py 结束
```

它怎样工作

你再一次看到了随机数字发生器的使用。这一次用户必须猜测脚本选择的数字。这个操作从脚本创建一系列从 1 到 100 的数字开始。当用户第一次键入猜测的数字时，在这个猜测数字和最后一个猜测数字之间没有不同之处，所以游戏要求另一个猜测数字。游戏获得它们之间差别的绝对值并告诉你猜测的是近还是远。如果你猜中了这个数字，游戏就会告知并祝贺你。

由 Joe Strout 提供，他是 CA 的 La Jolla 的 Salk Institute 中的科学软件开发者。

12.6 questor.py

实际上，这个脚本就像和你玩猜测游戏一样灵巧。

```
# questor.py 3/11/96 Joe Strout
# 定义一些常量以备将来之用

kQuestion = 'question'
kGuess = 'guess'

# 定义一个函数，提问是非题

def yesno(prompt):
    ans = raw_input(prompt)
    return (ans[0]=='y' or ans[0]=='Y')

# 在问题树中定义一个节点（问题或猜谜）
class Qnode:
    # 初始化方法
    def __init__(self, guess):
        self.nodetype = kGuess
        self.desc = guess

# 获取问题回答
    def query(self):
        if (self.nodetype == kQuestion):
            return self.desc + " "
        elif (self.nodetype == kGuess):
```

```
        return "Is it a " + self.desc + "? "
    else: return "Error: invalid node type!"

# 返回新节点, 给出的布尔响应
def nextnode(self, answer):
    return self.nodes[answer]

    # 将一猜谜节点转换为一问题节点, 并增加新条目, 给出问题、新的条目以及此条目的答案

def makeQuest( self, question, newitem, newanswer ):
    # 为新答案和旧答案创建新节点
    newAnsNode = Qnode(newitem)
    oldAnsNode = Qnode(self.desc)
    # 将此节点转换为一问题节点
    self.nodetype = kQuestion
    self.desc = question
    # 相应指定 yes 和 no 节点
    self.nodes = {newanswer:newAnsNode, not newanswer:oldAnsNode}

def traverse(fromNode):
    # 提问
    yes = yesno( fromNode.query() )
    #如果这是一个猜谜节点, 判断是否能到它的右部
    if (fromNode.nodetype == kGuess):
        if (yes):
            print "I'm a genius!!!"
            return
        # 如果不能到达右部, 返回节点
        return fromNode

    # 如果是一问题节点, 问另一个问题
    return traverse( fromNode.nextnode(yes) )

def run():
    # 以单独的猜谜节点开始
    topNode = Qnode('python')
    done = 0
    while not done:
```

```

# 问问题，直到结束
result = traverse( topNode )
# 如果结果有节点，需要增加一个问题
if (result):
    item = raw_input("OK, what were you thinking of? ")
    print "Enter a question that distinguishes a",
    print item, "from a", result.desc + ":"
    q = raw_input()
    ans = yesno("What is the answer for " + item + "? ")
    result.makeQuest( q, item, ans )
    print "Got it."
# 重复，直到完成
print
done = not yesno("Do another? ")
print
# 立即模式指令，用于执行拖拉或 execfile()
if __name__ == '__main__':
    run()
    print
    raw_input("press Return")
else:
    print "Module questor imported."
    print "To run, type: questor.run()"
    print "To reload after changes to the source, type: reload(questor)"
# questor.py 结束

```

它怎样工作

这个脚本明确地表达适应你的应答的问题并存储旧的回答。要完成这些工作，它需要一个问题树，由类 `Qnode` 构建它的节点。这个脚本存储在字典 `self.nodes` 中添加到树的每个节点。

类 `Qnode` 被使用过多次。例如，在下面的行中：

```
topNode = Qnode('python')
```

`Qnode` 作为 `topNode` 实例化并给出默认字符串 'python'，以便于当游戏开始时适用于脚本的某事呈现给用户。`newAnsNode` 和 `oldAnsNode` 也是使用 `Qnode` 的例子。

正像作者指出的一样，程序的所有手法都是临时的，因为当编写程序时，没有办法为次

日的游戏存储数据的拷贝。因此，你每次玩时必须重新构建 Questor。在游戏结束时编写一个文件例行程序为数据打开、附加并关闭文件是迷人的，当玩游戏的人再次启动游戏时就会读入这个例行程序。

附录 A Python 编程和已知故障常见问题解答 (FAQ)

本章要点:

- 构建 Python 和其他已知故障
- 用 Python 编程

学习技巧和材料内部最好的资源之一在 FAQ 中。FAQ 来自 python.org。你可以在 <http://www.python.org> 获得最新和完整的拷贝。

A.1 构建 Python 和其他已知故障

1.1 有测试集吗?

1.2 当运行测试集时, 我获得有关浮点运算的意见, 但是当我操作浮点运算时没有发现它们出了任何毛病。

1.3 在运行配置脚本之后的链接错误。

1.4 Python 解释程序抱怨传递到脚本的选项 (在脚本名字之后)。

1.5 当 SGI 上构建时, 试图运行 Python 来创建 `glmodule.c`, 但是 Python 还没有构建或安装。

1.6 我使用 `VPATH` 但是在源字典中构建了一些目标。

1.7 构建或与 GNU `readline` 库链接的故障。

1.8 关于旧版本 Linux 1.x 的套接字 I/O 方面的故障。

1.9 有关 Ultrix 原型方面的问题。

1.10 在平台 X 上构建 Python 的其他故障。

1.11 在 Linux 上如何配置动态装载。

1.12 在 Linux 2.0 (Slackware96) 上不能让共享模块工作。

1.13 当在 Linux 上共享模块时的故障。

1.14 在 Linux 上如何使用线程。

1.15 当和包含 C++ 代码的共享库链接时的错误。

1.16 我用激活的 `tkintermodule.c` 构建却得到 “Tkinter not found (没有找到 Tkinter)”。

1.17 我用 Tk4.0 构建, 但 Tkinter 抱怨 Tk 版本。

1.18 `_tkinter` 模块的编译或链接错误。

1.19 我为 Tcl/Tk 配置和构建 Python, 但是 “import Tkinter (输入 Tkinter)” 失败。

- 1.20 Tk 在 DEC Alpha 上不能正确工作。
- 1.21 几个普通的系统调用从 posix 模块中消失。
- 1.22 ImportError: 在 MS Windows 没有命名为字符串的模块。
- 1.23 在 SGI 上使用 gl 模块时存储内核信息。
- 1.24 在 MS-Windows 上构建 DLL 时 “Initializer not a constant (初启化算子不是常数)”。
- 1.25 在 Linux 上送往管道或文件上的输出消失。
- 1.26 在具有 libc5.4 的 Linux 中到处都是 Syntax Errors (语法错误)。
- 1.27 当使用 Tkinter 时, 在 Linux 上的 XIO 崩溃。
- 1.28 如何测试 Tkinter 是否正在工作?
- 1.29 有使 Python 解释程序的交互模式执行函数/变量名字完成的方法吗?
- 1.30 为什么 Python 不像共享库一样构建?
- 1.31 在 Solaris2.6 (SunOS5.6) 上用 GCC 的构建失败。
- 1.32 运行 “make clean”, 看来似乎保留了有问题的文件而导致随后的构建失败。

A.2 用 Python 编程

- 2.1 有源代码级的调试器吗, 如断点、单步运行等?
- 2.2 我能创建具有用 C 语言和 Python 实现的一些程序的对象类吗 (例如通过继承)?
(也可以用短语表达: 我能像基类一样使用内置类型吗?)
- 2.3 有适用于 Python 的 curses/termcap 包吗?
- 2.4 在 Python 中有与 C 语言的 onexit() 等效的吗?
- 2.5 当我在另一个函数内定义嵌套函数时, 嵌套函数看上去不能访问外部函数的局部变量。怎么办? 如何给嵌套函数传递局部数据?
- 2.6 如何在次序颠倒的序列中进行累加?
- 2.7 我的程序太慢, 如何给它加速?
- 2.8 当输入模块、然后编辑它并再次输入它 (进入同样的 Python 进程) 时, 所做的改动看上去没有发生。为什么?
- 2.9 如何找到当前模块名?
- 2.10 当模块作为脚本运行时我想在模块中执行一些附加的代码。如何发现我是否正在像脚本一样运行模块?
- 2.11 我试图从 Demo 字典中运行程序, 但它由于 ImportError 而失败: 没有命名的模块...; 发生了什么事?
- 2.12 我已经用 STDWIN 成功地构建了 Python, 但它找不到一些模块 (例如 stdwinevents)。
- 2.13 有哪些适用于 Python 的 GUI 工具包?

- 2.14 在 Python 中有到数据库包的接口吗？
- 2.15 用 Python 编写困惑的俏皮话是可能的吗？
- 2.16 有 C 语言的“?:”三元运算符的等价物吗？
- 2.17 我的类定义了 `_del_`，但是当我删除对象时它没有被调用。
- 2.18 我如何改变使用 `os.popen()` 或 `os.system()` 调用的程序的 shell 环境呢？
- 2.19 什么是类？
- 2.20 什么是方法？
- 2.21 什么是自身？
- 2.22 什么是非绑定方法？
- 2.23 如何从重载它的子类中调用在基类中定义的方法
- 2.24 如何不使用基类的名字从基类中调用方法？
- 2.25 如何组织代码使它轻松改变基类？
- 2.26 如何找到对象的方法或属性？
- 2.27 看上去似乎不能在 `os.popen()` 创建的管道上使用 `os.read()`。
- 2.28 如何从 Python 脚本中创建单独的二进制文件？
- 2.29 对 Python 来说 WWW 工具是什么？
- 2.30 如何用连接到输入和输出的管道运行了进程？
- 2.31 如果元组中有参数如何调用函数？
- 2.32 如何用 Emacs 使字体锁定方式适用于 Python？
- 2.33 有 `scanf()` 或 `sscanf()` 等价物吗？
- 2.34 等待 I/O 时，我能处理 Tk 事件吗？
- 2.35 如何用输出参数（按引用调用）编写函数？
- 2.36 请解释 Python 中局部和全局变量的规则。
- 2.37 如何让模块彼此相互地输入？
- 2.38 如何用 Python 拷贝对象？
- 2.39 如何用 Python 实现持久性对象？（持久性 \Rightarrow 自动地保存到磁盘并从磁盘再生。）
- 2.40 我试图使用 `_spam`，然后得到一个关于 `_SomeClassName_spam` 的错误。
- 2.41 如何删除文件？和另外一个文件问题。
- 2.42 如何修改 `urllib` 或 `httplib` 以支持 HTTP/1.1？
- 2.43 `compile()` 或 `exec` 中不可解释的语法错误。
- 2.44 如何将字符串转换为数字？
- 2.45 如何将数字转换为字符串？
- 2.46 如何拷贝文件？
- 2.47 如何检查对象是否为给定类的实例或它的子类？

- 2.48 什么是 delegation (授权)?
- 2.49 如何测试 Python 程序或组件?
- 2.50 多维列表 (数组) 被断开! 会给出什么?
- 2.51 我想做一个复杂的排序: 你能用 Python 做一个 Schwartzian 变换吗?
- 2.52 如何在元组和列表之间转换?
- 2.53 用 urllib 检索的文件包含前端无用信息, 看起来像 e-mail 报头。
- 2.54 如何得到一系列给定类的所有实例?
- 2.55 关于 regex.error 的正规表达式失败: 匹配失败。
- 2.56 得不到工作的信号处理器。
- 2.57 在函数中不能使用全局变量? 请帮助!
- 2.58 什么是负索引? 为什么 list.insert() 不使用它们?
- 2.59 如何按另一个列表的数值排序一个列表?
- 2.60 为什么 dir() 不能像文件和列表一样影响内置类型?
- 2.61 如何模仿 CGI 形式提交 (METHOD=POST)?
- 2.62 如果我的程序使用一个打开的 bsddb (或 anydbm) 数据库, 它就会被破坏。为何?
- 2.63 如何使 Python 脚本在 UNIX 上可执行?
- 2.64 如何从列表中删除副本?
- 2.65 Python 中有众所周知的 2000 年问题吗?
- 2.66 我想要将方法应用于对象序列的映射版本! 请帮助!
- 2.67 如何用 Python 产生随机数?
- 2.68 如何访问串行端口 (RS232)?
- 2.69 Py15 中 Tk-Buttons 的图像不正常?
- 2.70 math.py (socket.py、regex.py 等) 源文件在哪里?
- 2.71 如何从 Python 脚本中发送邮件?
- 2.72 在套接字的 connect() 中如何避免阻塞?
- 2.73 如何指定十六进制和八进制的整数?
- 2.74 如何每次得到一个单个按键?
- 2.75 如何在 Python 中重载构造函数 (或方法)?
- 2.76 如何从一个方法到另一个方法传递关键字参数?
- 2.77 应该使用什么模块帮助产生 HTML?
- 2.78 如何从 doc 字符串中创建文档?
- 2.79 如何读取 (或编写) 二进制数据?
- 2.80 在 Tkinter 中不能使用键绑定。
- 2.81 “import crypt (输入 crypt)” 失败。
- 2.82 有适于 Python 程序的编码标准或样式向导吗?

- 2.83 如何冻结 Tkinter 应用程序?
- 2.84 如何创建静态类数据和静态类方法?
- 2.85 `import('x.y.z')` 返回 `<module 'x'>`; 如何得到 `z`?
- 2.86 基本线程知识。
- 2.87 关闭 `sys.stdout` (`stdin`、`stderr`) 为什么不真正关闭它?
- 2.88 什么全局数值变化是线程安全的?
- 2.89 如何适当修改字符串?
- 2.90 如何传递关键字/可选参数/参数。

A.3 构建 Python 和其他已知故障

1.1 有测试集吗?

当然。在用“`make test` (进行试验)”构建之后你能运行它, 或者用下面的命令手动运行它:

```
import test.autotest
```

在 1.4 版本或更早的版本中, 使用:

```
import autotest
```

测试集不能测试 Python 的所有功能, 但它对于确认 Python 实际上是否正在工作是大有帮助的。

如果“`make test`”失败, 不要只把输出结果寄往新闻组——这样不会给调试问题提供足够的信息。相反, 要找出哪个试验失败了, 并从交互解释程序中手动运行那个测试。例如, 如果“`make test`”报告 `test_spam` 失败, 尝试以下这种交互式:

```
import test.test_spam
```

一般产生有更冗长的输出, 通过诊断它来调试问题。

1.2 当运行测试集时, 我获得有关浮点运算的意见, 但是当我操作浮点运算时没有发现它们出了任何毛病。

测试集偶尔会生成关于 C 浮点运算的语义的无根据的假定。直到有人提供一个更好的浮点测试集, 你必须注解不愉快的浮点测试并手动执行类似的测试。

1.3 在运行配置脚本之后的链接错误。

在配置改变之后运行“`make clean` (进行清除)”通常是有必要的。

1.4 Python 解释程序抱怨传递到脚本的选项 (在脚本名字之后)。

你可能和 GNU 的 `getopt` 连接在了一起, 举例说明, 经由——`liberty`。如果不是, 抱怨的原因是 GNU `getopt`, 和 System V `getopt` 和其他的 `getopt` 不同, 不考虑选项列表结尾的非选项。脚本的快速 (和兼容) 调整是给解释程序添加“`--`”, 如下所示:

```
#!/usr/local/bin/python --
```

你也可以使用这种交互式:

```
python - - script.py [options]
```

注意在 Python 发行版本 (在 Python/getopt.c 中) 中提供的正在工作的 getopt 实现, 但是不能自动使用。

1.5 当在 SGI 上构建时, 试图运行 Python 来创建 glmodule.c, 但是 Python 还没有构建或安装。

在安装程序中注解叙述 glmodule.c 的行, 并且最初不用 gl 构建 Python; 安装它并确信它在你的 \$PATH 中, 然后再次编辑 Setup 文件开启 gl 模块, 再次生成 Python。你不需要执行“make clean”; 你需要在 Modules 子字典中运行“make Makefile”(或只在顶层运行“make”)。

1.6 我使用 VPATH 但是在源字典中构建了一些目标。

在某些系统上 (例如, Sun), 如果目标已经存在于源字典中, 它就在那里创建而不是在构造字典中。这通常是因为你以前没有用 VPATH 构建。尝试在源字典中运行“make clobber”。

1.7 构建或与 GNU readline 库链接的故障。

考虑使用 readline2.0。一些提示:

你可以用 GNU readline 库改善交互式用户接口: 当交互式调用 Python 时, 会给出行编辑和命令的历史记录。在运行配置脚本之前, 你需要配置和构建 GNU readlin 库。它的原始资料不再和 Python 一起发行; 你可以从 GNU 镜像站点或它的主站点 (<ftp://slc2.ins.cwru.edu/pub/dist/readline-2.0.tar.gz>) 下载它们 (或使用更高级版本号——使用版本 1.x 不是所推荐的)。检查 Python 配置脚本选项“readline=DIRECTORY”, 其中 DIRECTORY 是你已创建的 readline 库的绝对路径。关于构建和使用 readline 库的一些提示: 在 SGI IRIX5 中, 你也许必须给 rldefs.h 添加下面的行:

```
#ifndef sigmask
#define sigmask(sig) (1L << ((sig)-1))
#endif
```

在大多数系统上, 你必须在几个源文件的顶端添加#include "rldefs.h", 并且如果你使用 VPATH 功能, 为了 foo 的几个数值必须给 Makefile 添加格式 foo.o:foo.c 的相依性。readline 库需要 termcap 库的使用。一个已知问题是它包含了导致 STDWIN 和 SGI GL 库冲突的入口点。STDWIN 冲突通过向 stdwin.h 文件增加一行“# define werase w_erase”就可以解决。GL 冲突能用 Python 配置脚本解决, 通过强制 termcap 库的静态版本的使用的 hack。检查新闻组 gnu.bash.bug news:gnu.bash.bug 可以得到关于 readline 库的特殊问题 (我没有读过这个组, 但是有人告诉我它就在 reading 故障的位置)。

1.8 关于旧版本 Linux 1.x 的套接字 I/O 方面的困难。

一旦你已经构建了 Python, 使用它在 Lib/linux1 字典下运行 regen.py 脚本。显然在某些

Linux 版本上发行的文件与系统头不匹配。

1.9 有关 Ultrix 原型方面的问题。

Ultrix cc 看上去似乎被中断——使用 gcc，或者把 config.h 编辑到 #undef HAVE_PROTOTYPES。

1.10 在平台 X 上构建 Python 的其他故障。

请把详细资料 e-mail 到 guido@cnri.reston.va.us，请提供尽可能多的详细资料。特别是，如果你没有说明计算机的类型和正在使用的操作系统（或版本），很难指出问题所在。如果你得到一个特殊的错误消息，也请把它 e-mail 到 guido@cnri.reston.va.us。

1.11 在 Linux 上如何配置动态装载。

只要 Linux 版本使用 ELF 对象格式，现在它就是自动的（所有最近的 Linux 版本都这样）。

1.12 在 Linux 2.0 (Slackware96) 上我不能让共享模块工作。

这是 Slackware96 版本中的一个故障。调整很简单：

确信从 /lib/libdl.so 到 /lib/libdl.so.1 有一个链接，以便于设置下面的链接：/lib/libdl.so->/lib/libdl.so.1 /lib/libdl.so.1 -> /lib/libdl.so.1.7.14

将 config.cache 文件删除、改动之后，如果要重新编译构建 Python，你可能必须重新配置一下脚本。

1.13 当在 Linux 上共享模块时的故障。

当你已经为静态链接构建了 Python，然后在 Setup 文件中启动 *shared* 时，就会发生这种故障。共享的库代码必须用 “-fpic” 编译。如果适用于模块的 .o 文件已经存在，并编译成静态链接时，你必须删除它或在 Modules 字典中进行 “make clean”。

1.14 在 Linux 上如何使用线程。

[Greg Stein]你需要有一个非常新的乃至比较好的 libc，获得 LinuxThreads-0.5 发行版本。注意如果你正常安装 LinuxThreads，你不应该需要指定字典为 -with-thread 设定开关。配置脚本找到它应该没有问题。为了确信每件东西都正确构建，使用 “make clean”，删除 config.cache，重新运行带有开关的配置，然后构建。

[Andy Dustman]在 glibc 系统上（如 RedHat 5.0+），LinuxThreads 被 POSIX 线程（-lpthread）舍弃。如果从一个早期 Red Hat 上进行升级，用 “rpm -e linuxthreads linuxthreads-devel” 删除 LinuxThreads。然后如同上述使用 -with-thread 运行配置。

1.15 当和包含 C++ 代码的共享库链接时的错误。

用 C++ 链接主 Python 二进制。在你的 C++ 编译器 Modules/Makefile 中改变 LINKCC 的定义。你也许必须稍微编辑 config.c，使它可用 C++ 进行编译。

1.16 我用激活的 tkintermodule.c 构建却得到 “Tkinter not found（没有找到 Tkinter）”。

Tkinter.py（注意大写字母 T）存在于 Lib 的子目录 Lib/tkinter 中。如果你正在使用默认模块搜索路径，可能没有启动 Modules/Setup 文件中定义 TKPATH 的行；如果使用环境变量 PYTHONPATH，将必须添加适当的 tkinter 子目录。

1.17 我用 Tk4.0 构建, 但 Tkinter 抱怨 Tk 版本。

有几个原因能导致这种情况发生。很可能有一个不能完全由 Tk4.0 安装 (注意在它的安装文件中添加 4.0) 删除的 Tk3.6 安装版本。你也许使 Tk3.6 支持库安装在了 Tk4.0 支持文件应该在的位置 (默认为 `/usr/local/lib/tk/`); 你也许用旧的 `tk.h` 头文件 (是的, 确定可以进行编译!) 编译了 Python; 实际上, 即使有 Tk4.0 你也用 Tk3.6 进行链接了。类似于 Tcl 7.4 与 Tcl7.3。

1.18 `_tkinter` 模块的编译或链接错误。

很可能, 在 Tcl/Tk 头文件 (`tcl.h` 和 `tk.h`) 和你正在使用的 Tcl/Tk 库 (举例来说, 在 `Setup` 文件中 “`-ltk8.0`” 和 “`-ltcl8.0`” 支持 `_tkinter` 的参数) 之间有一个版本不匹配。可能安装了几个版本的 Tcl/Tk 库, 但 `tcl.h` 和 `tk.h` 头文件只能有一个版本。如果库与头文件不匹配, 当链接模块或输入它时就会出现这个问题。幸运的是, 每个文件的版本号无疑都是一定的, 所以很容易发现。重新安装和使用最新版本通常能够修改这些问题。

(另外, 还要注意, 在编译未加补丁的针对 Tcl/Tk 7.6/4.2 的 Python 1.5.1 或更早的版本时, 你会得到一个 `Tcl_Finalize` 错误, 参见 <http://www.python.org/1.5/patches-1.5.1/> 上的 1.5.1 补丁页)。

1.19 我为 Tcl/Tk 配置和构建 Python, 但是 “`import Tkinter` (输入 `Tkinter`)” 失败。

很可能, 在 `Setup` 中忘记启用这一行: “`TKPATH=$(DESTLIB)/tkinter`”。

1.20 Tk 在 DEC Alpha 上不能正确工作。

你可能用 `gcc` 编译了 Tcl、Tk 或 Python。不要这样做。对于 64 位整数的这个平台, 知道 `gcc` 会产生破坏码。用标准 `cc` (和 OS 一起出现) 工作, 如果你仍然喜欢 `gcc`, 至少在把问题报告给新闻组或作者之前尝试用 `cc` 重新编译; 如果这样能解决问题, 就不用把问题报告给 `gcc` 开发者。(就我们所知, `gcc` 在其他平台上不存在这个问题——似乎不稳定性只局限在 DEC Alpha 上。)

而且 Tcl/Tk 还有一个 64 位的 `bugfix`, 请参阅 <http://grail.cnrireston.va.us/grail/info/patches/tk64bit.txt>。

1.21 几个普通的系统调用从 `posix` 模块中消失。

很可能, 所有由配置脚本运行的测试编译因为某些原因或另一些原因失败了。查看 `config.log` 看看是什么原因。一个常见的原因是指定目录为能包含 `libreadline.a` 文件的 `-with-readline` 选项。

1.22 `importError`: 在 MS Windows 没有命名为字符串的模块。

很可能, 你的 `PYTHONPATH` 环境变量应该被设置为下面的样子:

```
set PYTHONPATH=c:\python;c:\python\lib;c:\python\scripts
(假定 Python 安装在 c:\python 字典下)
```

1.23 在 SGI 上使用 `gl` 模块时存储内核信息。

在 `termcap` 和 `curses` 库中的入口点和 `GL` 库中的入口点之间有冲突。如果需要 GNU 的

`readline` 库，有大量针对 `termcap` 库的补丁，但在使用 `curses` 时往往不正常。最后，你不能构建同时包含 `curses` 和 `gl` 的 Python 二进制模块。

1.24 在 MS-Windows 上构建 DLL 时“Initializer not a constant(初始化算子不是常数)”。

在扩展模块的静态类型对象 `initializer` 也许会导致编译失败，带有像“`initializer not a constant`”这样的错误消息。Fredrik Lundh 在 Fredrik.Lundh@image.combitech.se 上解释。

当在 `MSVC` 下构建 DLL 时这种情况就会显露出来。有两种办法解决这种情况：像 C++ 一样编译这个模块，或者把你的代码改为如下所示：

```
static here PyTypeObject bstreamtype = {
    PyObject_HEAD_INIT(NULL) /* must be set by init function */
    0,
    "bstream",
    sizeof(bstreamobject),
    ...
    void
    initbstream()
    {
        /* Patch object type */
        bstreamtype.ob_type = &PyType_Type;
        Py_InitModule("bstream", functions);
        ...
    }
}
[cx]
```

1.25 在 Linux 上送往管道或文件上的输出消失。

有些人在它们交互式运行脚本时已经报告说，脚本运行得很顺利，若当它们重定向到管道或文件时，没有输出结果出现。

```
% python script.py
_some output_
% python script.py >file
% cat file
% # no output
% python script.py | cat
% # no output
%
```

没有人知道是什么导致了这种情况，但显然它是 Linux 的一个故障。大多数 Linux 用户

到现在也不能接受这种情况。

至少有一个重新安装 Linux (可能是更新版本) 和 Python 的人的报告摆脱了这种问题; 所以这也许就是解决方案。

1.26 在具有 libc5.4 的 Linux 中到处都是 Syntax Errors (语法错误)。

我已经在 Linux 系统上安装了 python1.4。当我尝试运行 import 语句时出现了下面的错误消息:

```
File "<stdin>", line 1
import sys
^
Syntax Error: "invalid syntax"
```

你亲自编译过它吗? 这通常是由于 libc 5.4x 和初期的 libc 版本不兼容造成的。尤其是, 用 libc5.4 编译的程序在安装 libc5.2 的系统上给出了不正确的结果, 因为 ctype.h 文件被破坏了。既然这样, Python 不能认出字符就是字母等。修改的方法是当构建你安装的二进制时安装使用的 C 库, 或者亲自编译 Python。当你做这些工作时, 确信由编译器使用的 C 库头文件与所安装的 C 库相匹配 (由 Martin V. Loewis 的应答改编而来)。

并且, (由 Andreas Jung 改编而来) 如果你已经升级为 libc5.4x, 而且问题仍然持续, 检查你的库路径寻找 libc 的旧版本。试图用 libs 和头文件清除更新的 libc, 然后尝试全部重新编译。

1.27 当使用 Tkinter 时, 在 Linux 上的 XIO 中崩溃。

当用 Linux 下的线程构建 Python 时, Tkinter 的使用能导致如下所示的崩溃:

```
>>> from Tkinter import *
>>> root = Tk()
X10: fatal 10 error 0 (Unknown error) on X server ":0.0"
after 45 requests (40 known processed) with 1 events remaining .
```

原因是默认 Xlib 没有用线程支持构建。如果能使线程重新构建, 这个问题就消失了。另一种可选择的方法是, 你可以不用线程重新构建 Python (首先 "make clean")。

1.28 如何测试 Tkinter 是否正在工作?

尝试下面的代码:

```
python
>>> import _tkinter
>>> import Tkinter
>>> Tkinter._test( )
```

这样可以弹出具有两个按钮的窗口, 一个是 "Click me", 另一个是 "Quit"。如果第一个语句 (import _tkinter) 失败, 你的 Python 安装可能没有配置支持 Tcl/Tk。在 UNIX 上, 如果已经安装了 Tcl/Tk, 在编辑 Modules/Setup 文件之后, 你必须重新构建 Python 以激活 tkinter

模块和 TKPATH 环境变量。

它也可能抱怨 Tcl/Tk 版本号不匹配,或缺少 TCL_LIBRARY 或 TK_LIBRARY 环境变量。要解决这些必须处理 Tcl/Tk 安装问题。

一个常见问题是 tcl.h 和 tk.h 的安装版本与 Tcl/Tk 库的安装版本不匹配;在装载共享库期间,这通常会导致链接程序错误或(当使用动态装载时)抱怨缺少符号。

1.29 有没有一个方法让交互模式的 Python 解释程序执行函数/变量名字完成?

(来自 Guido van Rossum 的记录)在 UNIX 上,如果你已经激活 readline 模块(例如,如果 Emacs 风格命令行编辑并且 bash 风格的历史记录可以工作),通过输入非正式文档的标准库模块“rlcompleter”可以添加这个模块。当完成一个简单标识器时,它完成了 `_main_` 的关键字、内置组件和全局变量;当完成 `NAME.NAME...` 时,它等价于最后的小数点并且完成它的属性提表达式。

这样,你可以进行“import string”,键入“string”,单击 completion 键两次,并且可看到由 string 模块定义的一系列名字。

为了作为 completion 使用 tab 键,调用:

```
readline.parse_and_bind("tab: complete")
```

你可以在 `~/pythonrc` 文件中放置这行代码,并把 PYTHONSTARTUP 环境变量设置为 `~/pythonrc`。无论什么时候交互式运行 Python,都将导致激活 completion。

注意(参见 `rlcompleter.py` 的 docstring 以得到更多信息):

- 如果发现具有 `__getattr__` 异常分支的对象, `NAME.NAME...` 形式的计算也许会导致代码定义的任意应用程序被执行。因为它是激活这个功能的应用程序(或用户)的职责,我认为这是可以接受的风险。更复杂的表达式(举例来说,函数调用或变址运算)都不被计算。
- GNU readline 也可以被内置函数 `input()` 和 `raw_input()` 使用,而且这些也受益于/忍受这个完整的功能。无疑地,一个交互式应用程序能从指定它自己的 completer (完成符)函数和使用 `raw_input()` 进行它的全部输入中受益。
- 当 stdin 不是 tty 设备时,GNU readline 永远不会被使用,并且这个模块(和 readline 模块)默默地处于非激活状态。

1.30 为什么 Python 不像共享库一样构建?

(这是一个 UNIX 问题;在 Mac 和 Windows 系统上,它是一个共享库。)让它在所有不同的平台上工作是一个恶梦。共享库的可移植性是一种痛苦。而且,我知道 GNU libtool——但它需要使用它的文件名转换等,而且我目前使用的版本,它需要所有 makefile 和 config 工具的完整和无条件重写工作。

尤其是,很少有应用程序能够嵌套在 Python 中——拥有已经将库共享的 Python 扩展更加常见。同样,糟糕的嵌套器通常想要全面控制它们使用的 Python 版本和配置,所以无论如何它们都不想使用标准共享库。因此,虽然使用有大量应用程序嵌套的 Python 以节省空

间的动机在理论上是好的，但我宁愿节省更多的实践时间（因此我给出共享库的低优先级）。

1.31 在 Solaris 2.6 (SunOS 5.6) 上用 GCC 的构建失败。

如果你已经将 Solaris 2.5 或 2.5.2 升级为 Solaris 2.6，但你没有升级 GCC 安装，编译也许就会失败，如下所示：

```
In file included from /usr/include/sys/stream.h:26,
      from /usr/include/netinet/in.h:38,
      from /usr/include/netdb.h:96,
      from ./socketmodule.c:121:
/usr/include/sys/model.h:32: #error "No DATAMODEL_NATIVE specified"
```

解决方案：为 Solaris 2.6 重新构建 GCC。你也许可以简单地重新运行 `fixincludes`，但是人们用这种方法可获得多方面的成功。

1.32 运行 “make clean”，看来似乎保留了有问题的文件而导致随后的构建失败。

相反，使用 “make clobber”。在中肯了你的构建和安装之后，使用 “make clean” 缩小 `source/build` 字典的大小。如果你已经尝试构建 Python 并且愿意再次构建，你应该使用 “make clobber”。它可以进行 “make clean” 并且也删除文件，例如从以前的构建中部分地构建的 Python 库。

A.4 用 Python 编程

2.1 有源代码级的调试器吗，如断点、单步运行等？

是的。检查模块 `pdb`。在 *Library Reference Manual* 中它是有文档的：`pdb.help()` 也打印文档。例如你可以通过使用 `pdb` 代码写自己的调试器。

Pythonwin 也有基于 `bdb` 上的可用的 GUI 调试器，能给断点涂上颜色并有少数很酷的功能（包括调试 non-Pythonwin 程序）。这个接口需要一些工作，但它还是很有趣的。在 <http://www.python.org/ftp/python/pythonwin/pwindex.html> 上可以找到参考。

为了以流行的 Data Display Debugger (DDD) 方式使用，Richard Wolff 创建了 `pdb` 的修改版本，称为 `Pydb`。`Pydb` 能在 <http://daikon.tuc.noao.edu/python/> 上找到，而且在 <http://www.cs.tu-bs.de/softech/ddd/> 上能找到 DDD。

2.2 我能创建有用 C 语言和 Python 实现的一些程序的对象类吗（例如通过继承）？（也可以用短语表达：我能像基类一样使用内置类型吗？）

不行，但你能很容易地创建在内置对象的周围承担包装的 Python 类，例如（对于字典来说）：

```
# 用户定义的类，与内置字典的功能基本相同。
class UserDict:
    def __init__(self): self.data={}
```

```

def _repr_(self): return repr(self.data)
def _cmp_(self, dict):
    if type(dict) == type(self.data):
        return cmp(self.data, dict)
    else:
        return cmp(self.data, dict.data)
def _len_(self): return len(self.data)
def _getitem_(self, key): return self.data[key]
def _setitem_(self, key, item): self.data[key] = item
def _delitem_(self, key): del self.data[key]
def keys(self): return self.data.keys()
def items(self): return self.data.items()
def values(self): return self.data.values()
def has_key(self, key): return self.data.has_key(key)

```

参见 Jim Fulton 的 `ExtensionClass` 得到允许你有超级类的一个机制的例子，在 Python 中，你可以从超级类中继承——用这种方法，你可以从 C 超级类（称为 `mixin`）中得到一些方法和从 Python 超级类或你的子类中得到一些方法。参见 <http://www.digicool.com/releases/ExtensionClass/>。

2.3 有适用于 Python 的 `curses/termcap` 包吗？

[Andrew Kuchling]标准 Python 发行版本和 `Modules/`子字典中的 `curses` 模块一些出现，虽然它不是默认被编译的。然而，那个模块只支持无格式 `curses`；关于它你不能使用像着色剂这样的 `ncurses` 功能（虽然它将和 `ncurses` 模块链接在一起）。

Oliver Andrich 有一个支持这个的功能的增强的模块；在 <http://andrich.net/python/selfmade.html#ncursesmodule> 有一个可用的版本。

2.4 在 Python 中有与 C 语言的 `onexit()` 等效的吗？

是的，如果你输入 `sys` 并分配一个函数为 `sys.exitfunc`，当你的程序退出时，它就会被调用。由未经处理的异常删除，或者（在 UNIX 上）接收 `SIGHUP` 或 `SIGTERM` 信号。

2.5 当我在另一个函数内定义嵌套函数时，嵌套函数看上去不能访问外部函数的局部变量。如何继续？如何给嵌套函数传递局部数据？

Python 没有任意嵌套作用域。当你需要创建一个需要访问一些局部可用的数据的函数时，创建容纳数据的新类并返回那个类的实例的方法，例如：

```

Class MultiplierClass:
    def _init_(self, factor):
        self.factor = factor
    def multiplier(self, argument):

```

```

        return argument * self.factor
def generate_multiplier(factor):
    return MultiplierClass(factor).multiplier
twice = generate_multiplier(2)
print twice(10)
# 输出: 20

```

下面的交互式解决方案使用默认参数, 例如:

```

def generate_multiplier(factor):
    def multiplier(arg, fact = factor):
        return arg*fact
    return multiplier
twice = generate_multiplier(2)
print twice(10)
# 输出: 20

```

2.6 如何在次序颠倒的序列中进行累加?

如果它是列表, 最快的解决方案是:

```

list.reverse( )
try:
    for x in list:
        "do something with x"
finally:
    list.reverse( )

```

当你在循环中时有一个不利情况, 这个列表被临时颠倒了。如果你不喜欢这样, 可以生成一个拷贝。除了实际上比其他解决方案快以外, 这样似乎花费很大:

```

rev = list[:]
rev.reverse( )
for x in rev:
    <do something with x>

```

如果它不是列表, 一个更普通但比较慢的解决方案是:

```

for i in range(len(sequence)-1, -1, -1):
    x = sequence[i]
    <do something with x>

```

一个更为精致的解决方案是定义一个担当序列的类, 并得到次序颠倒的元素 (解决方案应归功于 Steve Majewski):

```

class Rev:

```

```

def __init__(self, seq):
    self.forw = seq
def __len__(self):
    return len(self.forw)
def __getitem__(self, i):
    return self.forw[-(i + 1)]

```

现在，你可以简单地写：

```

for x in Rev(list):
    <do something with x>

```

可惜的是，由于方法调用开销，这个解决方案是所有解决方案中最慢的。

2.7 我的程序太慢，如何给它加速？

一般来说，它是一个难办的问题。有许多技巧可以加快 Python 代码；我考虑用 C 语言重写部分代码只作为最后的手段。要注意的一件事是函数和（特别是）方法调用是相当昂贵的；如果你定义了具有许多微小函数（不做比得到或设置实例变量或调用另一个方法更多的事情）的纯粹的 OO 接口，你可以考虑使用更直接的方法，例如直接访问实例变量。同样看一下标准模块“profile”（在 Library Reference 手册中有描述），使它查明你的程序花费大部分时间的地方成为可能（如果你有忍耐性——profiling 本身能按数量级减慢你的程序）。

记住你从其他编程经验中知道的许多标准优化试探（heuristics）也可以应用于 Python。例如，为了避免内核系统调用的开销，使用庞大的写命令而不用较小的写命令给输出设备发送输出结果就比较快。因此，用“one-shot”编写所有输出的 CGI 脚本，要显著地快于那些编写许多小片输出的脚本。

同样，务必在适当的地方使用“aggregate（集合）”操作。例如，“slicing（分片）”功能允许程序使用极为优化的 C 语言实现的单个记号，切开列表和其他序列对象。这样就会得到如下所示的同样的结果：

```

L2 = [ ]
for i in range(3):
    L2.append(L1[i])
it is much shorter and far faster to use
L2 = list(L1[:3]) # 如果 L1 是一个列表，则"list"是多余的。

```

注意 map() 函数，特别是以内置方法和内置函数方式使用时，它能成为一个方便的加速器。例如，为了使两个列表的元素成对在一起：

```

>>> map(None, [1,2,3], [4,5,6])
[(1,4), (2,5), (3,6)]

```

为了计算许多正弦：

```

>>> map(math.sin, (1,2,3,4))

```



```
[0.841470984808, 0.909297426826, 0.14112000806, -0.756802495308]
```

在这样的情况下，映射操作完成得非常快。

集合操作的其他例子包括标准字符串内置模块的 `join`、`joinfields`、`split` 和 `splitfields` 方法。例如，如果 `s1...s7` 是大的字符串，那么 `string.joinfields([a1,s2,s3,s4,s5,s6,s7], "")` 也许比较明显的 `s1+s2+s3+s4+s5+s6+s7` 更快，因为“summation（求和）”将计算许多子表达式，而连接域一次完成所有的复制。为了操纵字符串也要考虑正规表达式库和“substitution（替换）”操作 `String %元组` 和 `String %字典`。同样要务必使用 `list.sort` 内置方法排序，参见 FAQ 的 2.51 和 2.59 得到适当的高级应用实例——`list.sort` 搜索除极端情况外的所有其他排序技术是最好的。

在标准库和提供库及扩展中有许多其他集合操作可用。另一常用技巧是“把循环放进函数或方法中”。例如，假设有一个运行很慢的程序，并使用 `profiler (profile.run)` 来决定 Python 函数 `ff` 被调用许多次。如果你注意到 `ff`：

```
def ff(x):
    ...do something with x computing result...
    return result
```

趋向于在循环中被调用，如 (A)：

```
list = map(ff, oldlist)
```

或 (B)

```
for x in sequence:
    value = ff(x)
    ...do something with value...
```

那么你通常能通过把 `ff` 重写为如下所示，来消除函数调用开销：

```
def ffseq(seq):
    resultseq = []
    for x in seq:
        ...do something with x computing result...
        resultseq.append(result)
    return resultseq
```

并且重写 (A) 为：

```
list = ffseq(oldlist)
```

和重写 (B) 为：

```
for value in ffseq(sequence):
    ...do something with value...
```

其他单个调用 `ff(x)` 转换为具有很少开销的 `ffseq([x])[0]`。当然，这项技术不总是合适的，并且还有你得出其他变体。

你通过明确地存储函数或方法在 `local` 变量中查找的结果，能获得一些性能。如下所示循环：

```
for key in token:
    dict[key] = dict.get(key, 0) + 1
```

每次循环都要调用 `dict.get`。如果方法不能改变，更快的实现为：

```
dict_get = dict.get # 查询方法
for key in token:
    dict[key] = dict_get(key, 0) + 1
```

在编译时而不是在运行时默认参数能用于决定这个数值。这项操作只适用于在程序执行期间不能改变的函数和对象，例如代替：

```
def degree_sin(deg):
    return math.sin(deg * math.pi / 180.0)
```

用：

```
def degree_sin(deg, factor = math.pi/180.0, sin = math.sin):
    return sin(deg * factor)
```

因为这种技巧适用于不被改变的默认参数，它只能用于你不关心出现在你的用户的可能混乱的 API 时。要得到和最优化相关的秩序，参见 <http://www.python.org/doc/essays/list2str.html>。

2.8 当输入模块、然后编辑它并再次输入它（进入同样的 Python 进程）时，所做的改动看上去没有发生。为什么？

因为效率以及一致性的原因，在第一次输入模块时，Python 只读取模块文件（换句话说由许多模块组成的程序，输入同样的基本模块，多次地读取这个基本模块）。为了强制重读已改变的模块，可以这样做：

```
import modname
reload(modname)
```

警告——这项技术并不是百分之百地简单。特别是，像下面这样组成语句的模块：

```
from modname import some_objects
```

将继续用导入的对象的旧版本工作。

2.9 如何找到当前模块名？

通过看（预定义）全局变量 `__name__`，模块能找出它自己的模块名。如果你当作脚本运行时，它会有 `__main__` 值。

2.10 当模块作为脚本运行时我想在模块中执行一些附加的代码。如何发现我是否正在像脚本一样运行模块？

参见以前的问题。例如，如果在你的模块的最后一行放置下面的行，只有当你的模块像脚本一样运行时才调用 `main()`：

```
if __name__ == '__main__': main()
```

2.11 我试图从 Demo 字典中运行程序,但它由于 ImportError 而失败:没有命名的模块...;发生了什么事?

这很可能是你的系统上没有配置的可选模块(用 C 语言编写的)。用像“Tkinter”、“stdwin”、“gl”、“Xt”或“Xm”这样的模块尤其会发生这种事情。对于 Tkinter、STDWIN 和许多其他模块,参见 Modules/Setup.in 得到关于如何给 Python 添加这些模块的更多信息,如果它完全可能的话。有时,你必须首先 ftp 和构建另一个包(例如用于 Tkinter 的 Tcl 和 Tk)。有时,模块只在特定的平台上工作(例如 gl 只在 SGI 机器上工作)。

注意如果抱怨是关于“Tkinter”(大写字母 T)的,并且你已经配置了模块“tkinter”(小写字母 t),解决方案不是把 tkinter 重命名为 Tkinter,或者反之亦然。你的模块搜索路径可能存在一些错误。检查 sys.path 的数值。

对于和 X 相关的模块(Xt 和 Xm)你必须做更多的工作:它们当前不是标准 Python 发行版本的一部分。你必须下载一个 Extensions 的 tar 文件,也就是 <ftp://ftp.python.org/pub/python/src/X-extension.tar.gz> 和相关的指令。参阅下一个问题。

2.12 用 STDWIN 成功地构建了 Python,但它找不到一些模块(例如 stdwinevents)。

有命名为'stdwin'库字典的子字典,它应该在默认模块搜索路径中。为了这个目的在 Modules/Setup(.in)中有一行你必须激活它——可惜的是,在最新版本中它不靠近其他相关的 STDWIN 行,所以很容易忽略这一点。

2.13 有哪些适用于 Python 的 GUI 工具包?

根据你针对的平台,有几个这样的工具包。有一个到 Tcl/Tk 窗口小部件集合的简洁的面向对象接口,称为 Tkinter。这是标准 Python 发行版本的一部分而且被很好地支持——当构建 Python 时,你所需做的是构建和安装 Tcl/Tk 并激活 Modules/Setup 中的_tkinter 模块和 TKPATH 定义。这可能是最容易安装和使用并且最完整的窗口小部件集合。将来很可能标准 Python GUI API 将基于或至少看起来非常像 Tkinter 接口。要得到更多关于 Tk 的信息,包括到原始资料的指针,参见 Tcl/Tk 的主页 <http://www.scriptics.com>。目前,Tcl/Tk 可以完全地移植到 Mac 和 Windows 平台(只包括 NT 和 95);你需要 Python 1.4beta3 或更新版本和 Tk 4.1 或更新版本。

有一个到 X11 的接口,包括 Athena 和 Motif 窗口小部件集合(和几个单独的窗口小部件,像 Mosaic 的 HTML 窗口小部件和 SGI 的 GL 窗口小部件),可以从 <ftp://ftp.python.org/pub/python/src/X-extension.tar.gz> 上得到。由 Sjoerd Mullender 的站点 sjoerd@cw.nl 提供支持。

在 X11 接口的顶端有(最近再生的)vpApp 工具包,由 Per Spilling 提供,目前由 Sjoerd Mullender 的站点 sjoerd@cw.nl 维护。参阅 <ftp://ftp.cwi.nl/pub/sjoerd/vpApp.tar.gz>。

Mac 端口具有丰富和扩大的模块集,该模块集支持本地的 Mac 工具箱调用。参阅 Mac 端口的说明文档。参阅站点 <ftp://ftp.python.org/pub/python/mac>。由 Jack Jansen jack@cw.nl 支持。

由 Mark Hammond 的站点 MHammond@skippinet.com.au 支援的 NT 端口包括到 Microsoft Foundation Classes 的接口和一个主要用 Python 写成的 Python 编程环境。参见 <ftp://ftp.python.org/pub/python/pythonwin/>。

有一个基于称为 WPY 的 Microsoft Foundation Classes 模型的面向对象 GUI，由 Jim Ahlstrom 的站点 Jim@interet.com 支援。用 WPY 编写的程序运行不变而且带有 Windows NT/95、Windows 3.1（使用 win32s）和 UNIX（使用 Tk）的所见即所得的外观。Windows 和 Linux 的源代码和二进制版本可以在 <ftp://ftp.python.org/pub/python/wpy/> 上得到。

废弃的或少数解决方案：

有一个到 wxWindows 的接口。wxWindows 是用 C++ 编写的可移植 GUI 类库。它支持 XView、Motif 和 Windows 作为目标。也有一些适用于 Macs 和 CURSES 的支持。wxWindows 保持了基础的图形工具包的外表和感觉。对 wxPython 的支持（由 Harri Pasanen 的站点 pa@tekla.fi 提供）有较低的优先级。

只适用于 SGI IRIX，有到完整 GL（图形库——低层次但非常好的 3-D 能力）和 FORMS（由 Mark Overmars 在 GL 的顶端构建的按钮和滑块等包——可以从 <ftp://ftp.cs.ruu.nl/pub/SGI/FORMS/> 上下载）的接口。当 OpenGL 取代时，这可能也要被废弃了。

有一个到 STDWIN 的接口，适用于 Mac 和 X11 的独立平台、低层次的窗口（windowing）接口。这个接口完全不被支持并且很快就被废弃。STDWIN 的源代码在 <ftp://ftp.cwi.nl/pub/stdwin/>。（关于 STDWIN 2.0 的更多信息，请访问 Steven.Pemberton@cwi.nl——我想它也已经废弃了。）

有一个到 WAFE 的接口，一个到 X11 Motif 和 Athena 窗口小部件集合的 Tcl 接口。WAFE 在 <http://www.wu-wien.ac.at/wafe/wafe.html> 上。（在 FAQ 的早期版本中叙述的 Fresco 端口似乎不再存在。查询 Mark Linton。）

2.14 在 Python 中有到数据库包的接口吗？

在 ftp 服务器的 contrib 区中有它们的所有收集品。参见 <http://www.python.org/ftp/python/contrib/Database/>。

2.15 用 Python 编写困惑的俏皮话是可能的吗？

是的。参见下面的 3 个例子，Ulf Bartelt 的惯例：

小于 1000 的质数

```
print filter(None, map(lambda y: y * reduce(lambda x, y: x * y != 0,
map(lambda x, y: y % x, range(2, int(pow(y, 0.5) + 1))), 1), range(2, 1000)))
```

前 10 个 Fibonacci 数

```
print map(lambda x, f: lambda x, f: (x <= 1) or (f(x-1, f) + f(x-2, f)): f(x, f),
range(10))
```

Mandelbrot 集

```

print ((lambda Ru,Ro,Iu,Io,IM,Sx,Sy:reduce(lambda x,y:x+y,map(lambda y,
lu,Iu,Io,Io,KJ=Ru,Ro=Ro,Sy=Sy,I=lambda yc,Iu=Iu,Io=Io,Ru=Ru,Ro=Ro,
i=IM,Sx=Sx,Sy=Sy:reduce(lambda x,y:x+y,map(lambda x,xc=Ru,yc=yc,Ru=Ru,Ro=Ro,
i=I,Sx=Sx,F=lambda xc,yc,x,y,k,f=lambda xc,yc,x,y,k,f:(k<0)or (x*x+y*y>=4.0) or
1+f(xc,yc,x*x-y*y+xc,2.0*x*y+yc,k-1,f):f(xc,yc,x,y,k,f):chr(64+F(Ru+x*
(Ro-Ru)/Sx,yc,0,0,i)),range(Sx))):L(lu+y*(Io-Iu)/Sy),range(Sy))))
(-2.1,0.7,-1.2,1.2,30,80,24)
#      \_ _/      \_ _/      | | | _ _  屏幕的行数
#      V          V          | | _ _  列数
#      |          |          | _ _  最大的迭代数
#      |          | _ _ _ _ _ y 坐标范围
#      | _ _ _ _ _ x 坐标范围

```

不要在自己家里的主机上尝试，孩子！

2.16 有 C 语言的“?:”三元运算符的等价物吗？

没有直接的等价物。在许多情况下，你能用“a 和 b 或 c”模仿 `a?b:c`，但有一个缺陷：如果 b 为零（或空，或无——检测任何事情都为假）那么将选择 c 代替。在许多情况下，你可以通过查看代码证明这事不会发生（举例来说，因为 b 是一个常数或有一个永远不会为假的类型），但一般来说它是一个问题。

Tim Peters（他希望自己是 Steve Majewski）建议了下面的解决方案：`(a 和 [b] 或 [c])[0]`。因为 [b] 是永远不会为假的单个列表，所以错误路径永远不会发生；那么，把 [0] 应用整个事件得到你实际上想要的 b 或 c。不幸的是，当用 if 重写代码真的很困难，这种很少的情况下它才起作用。

2.17 我的类定义了 `__del__`，但是当我删除对象时它没有被调用。

有几个可能的原因。`del` 语句没有必要调用 `__del__`——它简单地递减对象的引用计数，并且如果达到零 `__del__` 就会被调用。

如果你的数据结构包含循环链接（举例来说，每个孩子有一个双亲点的树并且每个双亲有一列孩子），引用计数永远不会返回到零。你必须定义明确的 `close()` 方法删除那些指针。请永远不要直接调用 `__del__`——`__del__` 应该调用 `close()` 并且 `close()` 应该明确不止一次地对同一个对象进行调用。

如果对象曾经有一个函数的局部变量（或参数，实际是一样的事情）用以实现捕捉 `except` 语句的表达式，有机会是对象的引用仍然像栈跟踪中包含的一样存在于函数的栈帧中。通常，删除（或比较好的是，分配 `None` 给）`sys.exc_traceback` 就可以做到这一点。如果在交互式解释程序中为未经处理过的异常打印堆栈，改为删除 `sys.last_traceback`。

当解释程序存在时有能删除所有对象的代码，但是如果你的 Python 已经配置为支持线程，这个代码就不能调用（因为其他线程也许仍然是激活的）。你可以使用 `sys.exitfunc` 定义

自己的 `cleanup` 函数（参见问题 2.4）。

最后，如果你的 `__del__` 方法引发了一个异常，这个事件会被忽略。用 Python 1.4beta3 启动，当异常发生时，会向 `sys.stderr` 打印出警告消息。

2.18 我如何改变使用 `os.popen()` 或 `os.system()` 调用的程序的 shell 环境呢？

你必须使用 Python1.4 之前的版本或者在没有 `putenv()` 库函数的（罕见的）系统上。

在 Python1.4 之前，修改到子 shell 的环境被解释程序置于一边，因为看起来没有什么制定得很好的更轻便的方法来完成它（尤其是一些系统，有 `putenv()`，另一些有 `setenv()`，而且有一些根本什么也没有）。自从 Python1.4 版本以来，几乎所有的 UNIX 系统都有 `putenv()`，对 Win32 API 来说也一样，因此 `os` 模块被更新，以便 `os.environ` 的变化能够捕捉到，并且做到响应 `putenv()` 的调用。

2.19 什么是类？

类是通过执行类语句创建的特别的对象类型。类对象用作模板，创建类实例对象，包含数据结构和指定到数据类型的程序例程。

2.20 什么是方法？

方法是你通常为某些对象 `x` 像 `x.name`（参数...）一样调用的函数。这个术语适用于类和类实例的方法以及内置对象的方法。（后面有完全不同的实现和只共享它们的用 Python 代码的调用方式。）类的方法（和类实例）在类定义中作为函数定义。

2.21 什么是自身 `self`？

`self` 只是方法的第一个参数的传统名称——例如，在类定义中定义的函数。在定义出现的情况中，为类的某些实例 `x` 作为 `meth(self, a, b, c)` 定义的方法应该像 `x.meth(a, b, c)` 一样调用；调用的方法将考虑像 `meth(x, a, b, c)` 一样调用。

2.22 什么是非绑定方法？

非绑定法是类中定义的方法，它还没有被绑定在一个实例中。如果你请求的类属性有碰巧是一个函数，你会得到一个非绑定法。如果请求实例属性你会得到一个绑定法。绑定法知道它属于哪种实例并调用它自动支持实例；非绑定法只知道它的第一个参数想要的类（一个导出的类也可以）。调用非绑定法不是“magically（如魔法般地）”从上下文中导出第一个参数——你必须明确地提供它。

2.23 如何从重载它的子类中调用在基类中定义的方法？

如果你的类定义以“`class Derived(Base:...)`”开始，那么你能调用在 `Base`（或 `Base` 的基类）中定义的方法 `meth`，作为 `Base.meth(self, arguments...)` 来调用。在这，`Base.meth` 是非绑定法（参见前面的问题）。

2.24 如何不使用基类的名字从基类中调用方法？

真的，不要这样做！似乎你可以调用 `self.__class__.__bases__[0].meth(self, arguments...)`，但是当从你的类中导出双重导出法时就会失败：对它的实例来说，`self.__class__.__bases__[0]` 是你的类，不是它的基类——因此（假设你正在 `Derived.meth` 中从基类中调用方法）你将启

动一个递归 (recursive) 调用。

通常当你想要这么做时，你在忘记在 Python 中类是第一个类。在实例或子类级上你能“指向”想要授权操作的类。例如，如果你想要使用超级类的“glorp”操作，你能指向要使用的正确的超级类。

```
class subclass (superclass1, superclass2, superclass3):
    delegate_glorp = superclass2
    ...
    def glorp(self, arg1, arg2):
        .. subclass specific stuff ..
        self.delegate_glorp.glorp(self, arg1, arg2)
    -

class subsubclass(subclass):
    delegate_glorp = superclass3
    ..
```

然而，注意在子类中设置 `delegate_glorp` 为子类，会在 `subclass.delegate_glorp` 上导致无穷递归调用。小心！也许你会有很强的好奇心。但要考虑简化设计 (?)。

2.25 如何组织代码使它轻松改变基类？

你可以为基类定义一个别名，在类定义之前，为它分配真正的基类，并在整个类中使用别名。那么你必须改变的是分配给别名的值。偶然，如果你想要动态地（根据资源的可用性）决定使用的基类，这个技巧也是很方便的。例子：

```
BaseAlias = <real base class>
class Derived(BaseAlias):
    def meth(self):
        BaseAlias.meth(self)
    ..
```

2.26 如何找到对象的方法或属性？

这要根据对象的类型。对于定义的用户类的实例 `x` 来说，实例属性可以在字典 `x.__dict__` 中找到，并且由它的类定义的方法和属性可以在 `x.__class__.__bases__[i].__dict__` (在 `range(len(x.__class__.__bases__))` 中能到 `i`) 中找到。你将必须遍历基类的树来查找所有类方法和属性。

许多内置类型，但不是全部，在 `x.__methods__` 中定义了一系列它们的方法名，并且如果它们有数据属性，它们的名称也许可以在 `x.__members__` 中找到。然而，这只是一个规定。要得到更多信息，阅读标准（除了无正式文件的）模块 `newdir` 的原始资料。

2.27 看上去似乎不能在 `os.popen()` 创建的管道上使用 `os.read()`。

`os.read()` 是低级函数，采用文件描述符（小整数）。`os.popen()` 创建一个高级文件对象——用作 `sys.std{in,out,err}` 并由内置 `open()` 函数返回的同样类型。因而，从用 `os.popen()` 创建的管道 `p` 中读取 `n` 字节，你需要使用 `p.read(n)`。

2.28 如何从 Python 脚本中创建单独的二进制文件？

“Tools/freeze/” 中的 “freeze” 工具可以做你想做的事情。参见 README。通过递归地扫描你的源代码得到输入语句（两种形式）和在标准 Python 路径上以及源字典中查找模块（内置模块）。然后，它把用 Python 编写的模块“编译”成 C 代码（使用 `marshal` 模块能把数组初始化运算符变成代码对象），并创建只包含实际上在程序中使用的那些内置模块的定制 `config` 文件。然后，编译产生的 C 代码，并把它和 Python 解释程序的其余部分链接，来形成作像你的脚本一样的自含二进制文件（self-contained binary）。

如果你的脚本的文件名以 “.py” 结束，freeze 程序才能工作。

2.29 对 Python 来说 WWW 工具是什么？

参见 Library Reference Manual 中标题为 “Interent 和 WWW” 的文章。也有一个用 Python 编写的 Web 浏览器，称为 Grail（在本书的 CD-ROM 中）——见 <http://grail.cnri.reston.va.us/grail/>。

2.30 如何用连接到输入和输出的管道运行子进程？

使用标准 `popen2` 模块。例如：

```
import popen2
fromchild, tochild = popen2.popen2("command")
tochild.write("input\n")
tochild.flush()
output = fromchild.readline()
```

注意 `popen2` 中的故障：除非你的程序调用 `wait()` 或 `waitpid()`，结束的子进程永远不会被删除，最后因为子进程数目的限制对 `popen2` 的调用将会失败。用 `os.WNOHANG` 选项调用 `os.waitpid` 能阻止这件事；插入这样的调用的合适位置是在再次调用 `popen2` 之前。

在许多情况下，你实际上需要的是在命令中运行一些数据并得到返回的结果。除非数据在大小方面是无穷的，做这项工作最容易的（并且通常是最有效的！）方法是把它写到一个临时文件中，并用输入这样的临时文件运行这条命令。标准模块 `tempfile` 导出函数 `mktemp()`，产生唯一的临时文件名。

注意许多交互式程序（例如 `vi`）不能用代替标准输入和输出的管道很好地工作。你必须使用冒充的 `ttys`（“`ptys`”）代替管道。有一些在库模块 `pty.py` 中使用这些无正式文件的代码——我恐怕你也在这。

一个不同的答案是到 Don Libes 的 “expect” 库的 Python 接口。Python 扩展中与 `expect`

接口叫“expy”，并可以从<ftp://ftp.python.org/pub/python/contrib/System/>上得到。

像 expect 一样工作的纯 Python 解决方案是由 John Croix 提出的 PIPE。PIPE 的预先发行版本可以从<ftp://ftp.python.org/pub/python/contrib/System/>上得到。

一般来讲，这样做是不聪明的表现，因为你可以很容易地引起死锁，这将锁定你的进程而只能等待从子进程中的输出，而子进程也被锁定而等待输出。可能引发这种现象是因为父进程希望子进程比它输出更多的文本，或者可以通过将数据由于缺少刷新而粘到音频缓冲器引起。Python 父进程在它读取任何输出前当然可以明确地刷新数据，但是如果子进程是本地 C 程序，则它可以容易地写入一个并没有明确地刷新的输出中，甚至如果它是交互性的，则该刷新通常是自动的。

2.31 如果元组中有参数如何调用函数？

使用内置函数 `apply()`。例如，

```
func(1, 2, 3)
```

等价于：

```
args = (1, 2, 3)
```

```
apply(func, args)
```

注意 `func(args)` 是不同的——它用一个参数和元组 `args` 代替三个参数（整数 1、2 和 3）调用 `func()`。

2.32 如何用 Emacs 使字体锁定方式适用于 Python？

如果你正在使用 XEmacs 19.14 或更新版本，任意 XEmacs20、FSF Emacs 19.34 或任意 Emacs 20，如果你使用最新的 `python-mode.el` 字体锁定会自动为你工作。

如果你正使用 XEmacs 或 Emacs 的旧版本，需要在 `.emacs` 文件中放置下面的行：

```
(defun my-python-mode-hook()
  (setq font-lock-keywords python-font-lock-keywords)
  (font-lock-mode 1))
(add-hook 'python-mode-hook 'my-python-mode-hook)
```

2.33 有 `scanf()` 或 `sscanf()` 等价物吗？

不是这样的。对于简单的输入语法分析来说，最容易的途径通常是用 `string.split()` 把行分成间隔空格的单词，并用 `string.atoi()`、`string.atol()` 或 `string.atof()` 把十进制字符串转换为数字值。（Python 的 `atoi()` 为 32 位而且它的 `atol()` 为任意精度。）如果你想使用另一个间隔符而不是空格，可以使用 `string.splitfield()`（把它和 `string.strip()` 结合，可能会删除字符串周围的空格）。

要得到更多复杂的输入语法分析，正则表达式（见模块 `regex`）比 C 语言的 `sscanf()` 更适合而且更强大。有一个由 Steve Clift 提供的仿真 `sscanf()` 的模块；见站点：<http://www.python.org/ftp/python/contrib/Misc/sscanfmodule.c> 的 `contrib/Misc/sscanfmodule.c` 页。

2.34 等待 I/O 时，我能处理 Tk 事件吗？

是的，你甚至不需要线程！但是你必须稍微重构 I/O 代码。Tk 有 Xt 的 `XtAddInput()` 调用的等价物，当 I/O 可能是文件描述符时，允许你注册从 Tk 主循环中调用的回调函数。下面是你所需要的：

```
from Tkinter import tkinter
tkinter.createfilehandler(file, mask, callback)
```

文件也许是 Python 文件或套接字对象（实际上，具有 `fileno()` 方法的任何东西），或者整数文件描述符。`mask` 是常数 `tkinter.READABLE` 或 `tkinter.WRITABLE` 的其中之一。`callback` 可以如下调用：

```
callback(file, mask)
```

当你做完时要取消注册 `callback`，使用：

```
tkinter.deletefilehandler(file)
```

注意因为你不知道有多少字节可用于读取，所以不能使用 Python 文件对象的 `read` 或 `readline` 方法，因为这些方法坚决要求读取预定义数目的字节。对于套接字来说，`recv()` 或 `recvfrom()` 方法可以很好地工作；对于其他文件，使用 `os.read(file.fileno(), maxbytecount)`。

2.35 如何用输出参数编写函数（引用调用）？

[Mark Lutz]要记住的是参数在 Python 中通过分配赋值来传递。因为分配赋值只创建对象的引用，在调用程序中的参数名和被调用者之间没有别名，所以本质上没有引用调用。但你可以用许多方法模拟它：

（1）通过使用全局变量：但你不会使用它。

（2）通过传递可变的（适当可改变的）对象：

```
def func1(a):
    a[0] = 'new-value'      # 'a' 引用可变的列表
    a[1] = a[1] + 1         # 改变共享的对象
args = ['old-value', 99]
func1(args)
print args[0], args[1]     # 输出：新值 100
```

（3）通过返回元组，容纳参数的最后数值：

```
def func2(a, b):
    a = 'new-value'        # a 和 b 是本地名称
    b = b + 1              # 指定新对象
    return a, b            # 返回新值
x, y = 'old-value', 99
print x, y                # 输出：新值 100
```

（4）从 Python 的对象模型中得出的其他思想。例如，它可以用可变字典更清楚地传递：

```
def func3(args):
    args['a'] = 'new-value'      # args 是可变字典
    args['b'] = args['b'] + 1    # 在适当的位置改变它
args = {'a': 'old-value', 'b': 99}
func3(args)
print args['a'], args['b']
```

(5) 或者用类实例捆扎数值:

```
class callByRef:
    def __init__(self, **args):
        for (key, value) in args.items():
            setattr(self, key, value)
    def func4(args):
        args.a = 'new-value'     # args 是一个可变的 callByRef
        args.b = args.b + 1      # 在适当的位置改变它
        args = callByRef(a='old-value', b=99)
        func4(args)
    print args.a, args.b
```

但是, 没有更好的理由用这种错综复杂的操作。

2.36 请解释 Python 中局部和全局变量的规则。

[Ken Manheimer]在 Python 中, 过程变量是隐含的全局变量, 除非它们在程序块的任何地方被分配外。在那样情况下, 它们是隐含的局部变量, 并且你需要明确地声明它们为全局变量。

虽然起先有点奇怪, 一会的考虑对这种情况进行了解释。一方面, 标记性全局变量的需求对不可预测的边界影响提供了防护; 另一方面, 如果所有的全局引用者需要全局变量, 你将一直使用全局变量。因此, 你必须像全局变量一样声明对内置函数或导入模块的组件的每次引用。这样将战胜用于标识副作用的全局声明的有效性。

2.37 如何让模块彼此相互地输入?

Jim Roskind 推荐如下所示进入每个模块:

首先, 全部导出 (像全局变量、函数和不需要导入基类的类)。

其次, 所有输入语句。

最后, 所有激活代码 (包括从导入数值初始化的全局变量)。

2.38 如何用 Python 拷贝对象?

在 Python 中没有一般拷贝操作: 然而, 大多数对象类型都有一些创建复制的方法。这是大多数普通对象如何创建复制的方法: 对于不可改变的对象 (数字、字符串、元组), 复制是没有必要的因为它们的值不能改变。对于列表 (和一般是可改变的序列类型), 复制由

表达式 `l[:]` 创建。对于字典，下面的函数返回一个复制：

```
def dictclone(o)
    n = {}
    for k in o.keys(): n[k] = o[k]
    return n
```

最后，对于一般对象“copy”模块为拷贝对象定义了两个函数。`copy.copy(x)`像前面的规则显示的那样返回拷贝。`copy.deepcopy(x)`也拷贝复合对象的元素。参见 *Library Reference Manual* 中关于这个模块的部分。

2.39 如何用 Python 实现持久性对象？（持久性==自动地保存到磁盘并从磁盘再生。）

库模块“pickle”现在用非常一般的方法解决了这个问题（虽然你仍然不能像打开文件、套接字或窗口一样存储它），而且库模块“shelve”使用 pickle 和(g)dbm 创建包含任意 Python 对象的持久性映象。为了尽可能好地执行性能，也要查找相对新的 cPickle 模块的最新版本。

做这件事的比较笨拙的方法是使用 pickle 的小同胞，marshal。marshal 模块提供了把非周期性的基本 Python 类型存储为文件和字符串的非常快的方法，并再次返回。虽然 marshal 不做像存储实例或处理适当地共享引用那样的特别事件，但它运行得极快。例如，装载半兆字节的数据花费的时间少于 1/3 秒（在某些机器上）。这通常胜于做比较复杂和一般的事情，例如用 pickle/shelve 使用 gdbm。

2.40 我试图使用 `_spam`，然后得到一个关于 `_SomeClassName_spam` 的错误。

加插两根下划线的变量是“mangled”，提供了定义类私有变量的简单但有效的方法。参见 Python Tutorial 中的“New in Release 1.4”。

2.41 如何删除文件？和另外的文件问题？

使用 `os.remove(filename)`或 `os.unlink(filename)`；要查看文档，见库手册的 `posix` 部分。它们是相同的——`unlink()`是这个函数的简单的 UNIX 名字。在 Python 的早期版本中，只有 `os.unlink()`是可用的。

要删除一个目录，使用 `os.rmdir()`；使用 `os.mkdir()`创建目录。要重命名文件，使用 `os.rename()`。要截取一个文件，使用 `f = open(filename, "r+")`打开它，并使用 `f.truncate(offset)`截取；默认地偏移为当前搜索位置。（“r+”模式打开文件进行读取和写入。）也有适用于用 `os.open()`打开的文件的 `os.ftruncate(fd, offset)`——只用于高级 UNIX 文件系统。

2.42 如何修改 `urllib` 或 `httplib` 以支持 HTTP/1.1？

给 Python 1.4 `httplib.py` 应用下面的修补程序：

```
41c141
< replypat = re.sub.gsub('\\.', '\\\\\\.', HTTP_VERSION) + \
...
> replypat = re.sub.gsub('\\.', '\\\\\\. ', 'HTTP/1.[0-9]+' ) + \
```

2.43 compile()或 exec 中不可解释的语法错误。

当语句块（和表达式相反）由 compile()、exec 或 execfile()编译时，它必须用一个新行来结束。在某些情况下，当锯齿状的程序块中的源代码结束时，它似乎至少需要两个新行。

2.44 如何将字符串转换为数字？

对于整数，使用内置 int()函数，例如 int('144') == 144。类似地，long()从字符串转换为长整型，例如 long('144') == 144L；float()转换为浮点型，例如 float('144') == 144.0。

注意这些操作仅限于十进制译码，以致于 int('0144') == 144 和 int('0x144')会引发 ValueError。

为了得到较大的灵活性，或在 Python 1.5 之前，导入模块 string 并为整数使用 string.atoi()函数，为长整数使用 string.atol()或者为浮点数使用 string.atof()。例如，string.atoi('100',16) == string.atoi('0x100',0) == 256。参见库参考手册部分得到字符串模块的更多详细资料。

当你用内置函数 eval()代替那些函数中的任意一个时，不推荐这样做，因为有人会传递给你一个 Python 表达式，带有意料不到的副作用（像重新格式化你的磁盘）。

2.45 如何将数字转换为字符串？

例如，为了转换数字 144 为字符串'144'，使用内置函数 repr()或备份引用符号（它们是等价物）。如果你想要十六进制或八进制表示，分别使用内置函数 hex()或 oct()。为了特别的格式，在字符串上使用%操作符，就像 C printf 格式。例如 "%04d" %144 得到'0144'，并且 "%3f" % (1/3.0) 得到'0.333'。参见库参考手册以得到详细资料。

2.46 如何拷贝文件？

通常会这样做：

```
infile = open("file.in", 'rb')
outfile = open("file.out", "wb")
outfile.write(infile.read())
```

然而，对于巨大的文件，你想要成碎块地读取/写入（或者你必须），并且如果你向深入挖你也许会找到其他技术问题。

可惜的是，没有平台能完全独立地应答。在 UNIX 上，你能使用 os.system()来激活“cp”命令（见你的 UNIX 手册查看如何调用）。在 DOS 或 Windows 上，使用 os.system()调用“COPY”命令。在 Mac 上，使用 macostools.copy(srcpath, dstpath)，它也将拷贝资源分支和 Finder 信息。

还有 shutil 模块，包含实现拷贝循环的 copyfile()函数；但在 Python 1.4 和它的早期版本中，用文本模式打开文件，甚至在 Python 1.5 中，它仍然不能足够好地应用于 Macintosh：它没有拷贝资源分支和 Finder 信息。

2.47 如何检查对象是否为给定类的实例或它的子类？

如果正在由 scratch 产生类，有适当的面向对象类型编程也更好——代替基于组成员上做不同的事情，为什么不使用方法和用不同的类定义不同的方法？

然而，有一些合法的情况需要你测试组成员。在 Python 1.5 中，你可以使用内置函数 `isinstance(obj,cls)`。

下面的途径用于早期的 Python 版本：

一个不显著的方法是像异常情况一样引发对象，并尝试用你测试的类捕捉异常情况：

```
def is_instance_of(the_instance, the_class):
    try:
        raise the_instance
    except the_class:
        return 1
    except:
        return 0
```

这项技术也用于从类的集合中区分 “subclassness”：

```
try:
    raise the_instance
except Audible:
    the_instance.play(largo)
except Visual:
    the_instance.display(gaudy)
except Olfactory:
    sniff(the_instance)
except:
    raise ValueError, "dunno what to do with .this!"
```

用于异常捕捉类或子类成员的事实。

不同的途径是测试类属性的出现，推测给定类的唯一性。例如：

```
class MyClass:
    ThisIs MyClass = 1
-
def is_a_MyClass(the_instance):
    return hasattr(the_instance, 'ThisIsMyClass')
```

这个版本比内置函数容易，而且可能很快（`inlined` 的确很快）。缺点是其他人能欺骗它：

```
class IntruderClass
    ThisIsMyClass = 1 # 伪装为 MyClass
-
```

但这也许会被看作一种功能（无论如何，在 Python 中有足够的其他方法欺骗它）。另一个缺点是类必须为成员测试作准备。如果你不为类“控制源代码”，修改类支持可测试性也

许是不可行的。

2.48 什么是 delegation (授权)?

授权指一种 Python 程序员在特别的情况下实现的面向对象技术。考虑下面代码:

```
from string import upper
class UpperOut:
    def __init__(self, outfile):
        self.__outfile = outfile
    def write(self, str):
        self.__outfile.write( upper(str))
    def __getattr__(self, name):
        return getattr(self.__outfile, name)
```

这里, 调用基础的 `self.__outfile.write` 方法之前, `UpperOut` 类重定义 `write` 方法为把参数字符串转换为大写字母, 但所有的其他方法都授权给基础的 `self.__outfile` 对象。授权是通过“魔术般的” `__getattr__` 方法实现的。请查看语言参考, 以得到关于这个方法的使用的更多信息。

注意授权能得到技巧方面的更一般的情况。特别是当属性必须被设置以及得到的类也必须定义 `__setattr__` 方法时, 要小心做这些事。

`__setattr__` 的基本实现粗略地等价于下面所示:

```
class X:
    -
    def __setattr__(self, name, value):
        self.__dict__[name] = value
    -
```

为了自身不会导致无穷递归, 大多数 `__setattr__` 的实现必须修改 `self.__dict__`, 来存储局部状态。

2.49 如何测试 Python 程序或组件?

首先, 它有助于编写程序以致于容易地通过使用好的模块化设计测试。尤其是, 你的程序几乎应该拥有其他函数或类方法中封装的全部功能——有时, 也有令人惊奇和高兴的使程序快速运行的效果 (因为局部变量访问比全局变量访问更快)。此外, 程序应该避免依赖于可变的全局变量, 因为这样做会使测试更加困难。你的程序的 “global main logic (全局主逻辑)” 也许可以像下面这样简单在你的程序的主模块的底部。

```
if __name__ == "__main__":
    main_logic()
```

一旦你的程序做为函数和类行为的集合组织时, 你应该编写实行行为的测试函数。测试程序组应该和自动测试序列的每个模块联合起来。听起来似乎有许多工作, 但因为 Python

是如此简洁和灵活所以它令人惊奇地容易。你可以通过编写和“产品代码”并联的测试函数，使编码更令人愉快和有趣，因此这会使它容易找到故障甚至设计初期的缺陷。

“支持模块”不会成为程序的主模块，也许包括调用模块的自身测试的“测试脚本解释”。

```
if __name__ == "__main__":  
    self_test()
```

当外部接口通过使用 Python 实现的“fake”接口不可用时，甚至对复合外部接口有影响的程序也可能会被测试。为了“fake”接口的例子，下面的类定义（部分）了“fake”文件接口：

```
import string  
testdata = "just a random sequence of characters"  
class FakeInputFile:  
    data = testdata  
    position = 0  
    closed = 0  
    def read(self, n=None):  
        self.testclosed()  
        p = self.position  
        if n is None:  
            result= self.data[p:]  
        else:  
            result= self.data[p: p+n]  
        self.position = p +len(result)  
        return result  
    def seek(self, n, m=0):  
        self.testclosed()  
        last =len(self.data)  
        p = self.position  
        if m==0:  
            final=n  
        elif m==1:  
            final=n+p  
        elif m==2:  
            final=len(self.data)+n  
        else:  
            raise ValueError, "bad m"
```



```

        if final<0:
raise IOError, "negative seek" .
        self.position = final
def isatty(self):
    return 0
def tell(self):
    return self.position
def close(self):
    self.closed = 1
def testclosed(self):
    if self.closed:
        raise IOError, "file closed"
Try f=FakeInputFile() and test out its operations.

```

2.50 多维列表 (数组) 被断开! 会给出什么?

你很可能想尝试生成如下所示的多维数组。

```
A = [[None] * 2] * 3
```

这样会使包含 3 个引用的长度为 2 的同一个列表。改变一行内容在所有行中都有显示, 这可能不是你想要的。下面的工作会更好:

```

A = [None]*3
for i in range(3)
    A[i] = [None] * 2

```

这样会产生包含三个不同的长度为 2 的列表。

如果你感觉古怪, 也可以用下面的方法来完成:

```

w, h = 2, 3
A = map(lambda i, w=w: [None] * w, range(h))

```

2.51 我想完成一个复杂的排序: 你能用 Python 生成一个 Schwartzian 变换吗?

是的, 并且用 Python 你只需编写一次:

```

def st(List, Metric):
    def pairing(element, M =Metric):
        return (M(element), element)
    paired = map(pairing, List)
    paired.sort()
    return map(stripit, paired)
def stripit(pair):
    return pair[1]

```

这项技术，归功于 Randal Schwartz，通过映射每个元素到它的“排序值”来排序列表的元素。例如，如果 `L` 是一列字符串，那么：

```
import string
Unsorted = st(L, string.upper)
def intfield(s):
    return string.atoi( string.strip(s[10:15]) )
Isorted = st(L, intfield)
```

`Unsorted` 给出了排序的 `L` 的元素，好像它们都是大写字母，并且 `Isorted` 给出了通过在字符串片断中出现的整数值排序的 `L` 的元素，字符串片断从 10 的位置开始到 15 的位置结束。注意 `Isorted` 也可以通过下面的代码行计算：

```
def Icmp(s1, s2):
    return cmp( intfield(s1), intfield(s2) )
Isorted = L[:]
Isorted.sort(Icmp)
```

但是因为这个方法对于 `L` 的每个元素都计算许多次 `intfield`，所以它比 Schwartzian 变换慢。

2.52 如何在元组和列表之间转换？

函数 `tuple(seq)` 以同样的顺序用同样的项目把任何序列转换为元组。例如，`tuple([1,2,3])` 可以得到 `(1,2,3)`，而且 `tuple('abc')` 得到 `('a','b','c')`。如果参数是元组，它不能生成拷贝但返回同样的对象，所以当你不能确信对象已经是元组时，对于调用 `tuple()` 的价值不大。

函数 `list(seq)` 以同样的顺序用同样的项目把任何序列转换为列表。例如，`list((1,2,3))` 得到 `[1,2,3]`，并且 `list('abc')` 得到 `['a','b','c']`。如果参数是一个列表，它只是像 `seq[:]` 那样生成拷贝。

2.53 用 `urllib` 检索的文件包含前端无用信息，看起来像 e-mail 头标。

服务器使用 HTTP/1.1；Python 1.4 中的 `httplib` 只认识 HTTP/1.0。参见问题 2.42 中的修补程序。

2.54 如何得到一类给定类的所有实例？

Python 不能跟踪一个类（或者内置类型）的所有实例。你能设计类的构造函数类跟踪所有的实例，但除非你非常聪明，这里存在一个实例永远不会被删除的缺陷，因为你的类的所有实例都保持了对它们的引用。

（这种技巧有规则地检查你所保持的实例的引用计数，并且如果引用计数低于确定的级别，从列表中删除它。决定级别是很棘手的——它一定比 1 大。）

2.55 关于 `regex.error` 的正则表达式失败：匹配失败。

这通常是由太多的回溯导致的；正则表达式引擎有容纳最多 4000 个回溯点的固定尺寸堆栈。由 “.” 匹配的每个字符占了一个回溯点，因此即使像下面这样的简单搜索：

```
regex.match('.*x', "x"*5000)
```

也将失败。

在 Python 1.5 中介绍的 `re` 模块中将会修正这个问题；参考 Library Reference 部分中关于 `re` 的更多信息。

2.56 得不到工作的信号处理器。

最普通的问题是信号处理器用错误的参数列表来声明。它被像下面这样调用：

```
handler(signum, frame)
```

所以它应该用两个参数来声明：

```
def handler(signum, frame):
```

```
...
```

2.57 在函数中不能使用全局变量？请帮助！

你是像下面这样做的吗？

```
x = 1 # 创建一个全局变量
```

```
def f():
```

```
    print x # 试图打印这个全局变量
```

```
...
```

```
    for j in range(100):
```

```
        if q>3
```

```
            x=4
```

如果你这样做，由于“`x=4`”的赋值，在 `f` 中对 `x` 的所有引用都是局部变量，而不是全局变量。在函数中分配的任何变量对函数来说都是局部变量，除非它被声明为全局变量。因此，“`print x`”企图打印一个未初始化的局部变量，将会引发一个 `NameError`。

2.58 什么是负索引？为什么 `list.insert()` 不使用它们？

Python 序列可以用正数和负数索引。正数 0 是第一个索引，1 是第二个索引，诸如此类。对于负数来说，-1 是最后一个索引，-2 是倒数第二个索引（靠近最后一个），诸如此类。想象一下，`seq[-n]` 同 `seq[len(seq)-n]` 是一样的。

使用否定的索引可以非常便利。例如，如果字符串行以新行结束，那么 `Line[:-1]` 就是除了新行外的所有行。

遗憾的是，内置的列表方法 `L.insert` 看不到负索引。这一特征可以看作是一个错误，但是既然现有的程序都依赖于这个特征，那么它可能会永远存在下去。`L.insert` 将负索引插入到列表头部，要想获得“正确”的负索引，请使用 `L[n:n] = [x]` 来替换 `insert` 方法。

2.59 如何按另一个列表的数值排序一个列表？

你能排序元组的列表。

```
>>> list1 = ["what", "I'm", "sorting", "by"]
```

```
>>> list2 = ["something", "else", "to", "sort"]
```

```
>>> pairs = map(None, list1, list2)
```

```
>>> pairs
[('what', 'something'), ('I'm', 'else'), ('sorting', 'to'), ('by', 'sort')]
>>> pairs.sort()
>>> pairs
[('I'm', 'else'), ('by', 'sort'), ('sorting', 'to'), ('what', 'something')]
>>> result = pairs[:]
>>> for i in xrange(len(result)): result[i] = result[i][1]
...
>>> result
['else', 'sort', 'to', 'something']
```

注意 “I’m” 排在 “by” 之前，因为按照 ASCII 的顺序大写字母 “I” 在小写字母 “b” 的前面出现。参阅 2.51。

2.60 为什么 dir() 不能像文件和列表一样影响内置类型？

它将会有——并且从 Python 1.5 开始（目前正在开发中）。使用 Python 1.4，通过查找它的 `__methods__` 属性能查明给定的对象支持哪种方法：

```
>>> List = [ ]
>>> List.__methods__
['append', 'count', 'index', 'insert', 'remove', 'reverse', 'sort']
```

2.61 如何模仿 CGI 形式的提交 (METHOD=POST)？

我喜欢检索 Web 页，它是张贴 (POST) 一种表单的结果。有现存的代码可以让我容易地完成这项工作吗？是的。这里有一个使用 `httplib` 的简单的例子。

```
#!/usr/local/bin/python
import httplib, sys, time
### 建立查询字符串
qs = "First=Josephine&MI=Q&Last=Public"
### 连接并发送服务器路径
httpobj = httplib.HTTP('www.some-server.out-there',80)
httpobj.putrequest('POST', '/cgi-bin/some-cgi-script')
### 现在生成 HTTP 标题的其他部分
httpobj.putheader('Accept', '/*/*')
httpobj.putheader('Connection', 'Keep-Alive')
httpobj.putheader('Content-type', 'application/x-www-form-urlencoded')
httpobj.putheader('Content-length', '%d' % len(qs))
httpobj.endheaders()
httpobj.send(qs)
```

查找到服务器在响应中的内容

```
reply, msg, hdrs = httpobj.getreply()
if reply != 200:
    sys.stdout.write(httpobj.getfile().read())
```

注意一般对于“url encoded posts (编码的 url 张贴)” (默认) 查询字符串一定是“quoted (引用)”; 例如, 当它们存在名字或数值时, 改变等号并间隔编码形式。使用 `urllib.quote` 来执行这个引用。例如, 发送 `name="Guy Steele,Jr."`:

```
>>> from urllib import quote
>>> x = quote("Guy Steele, Jr.")
>>> x
'Guy%20Steele,%20Jr.'
>>> query_string = "name="+x
>>> query_string
'name=Guy%20Steele,%20Jr.'
```

2.62 如果我的程序使用一个打开的 bsddb (或 anydbm) 数据库, 它就会被破坏, 为什么?

用 `bsddb` 模块 (通常用 `anydbm`, 以后择优使用 `bsddb` 模块) 写存取打开的数据库必须明确地使用数据库的 `close` 方法关闭。基础的 `libdb` 包高速缓存数据库内容, 需要转换成磁盘形式并进行写操作, 与在内核的写入缓冲区已经有了磁盘位的正规打开文件不同, 它们只能通过程序退出的内核来转储。

在程序崩溃之前, 如果已经初始化一个新的 `bsddb` 数据库但没有写入任何内容, 你通常会以零长度文件来结束, 并在下一次文件打开时遭遇一个异常。

2.63 如何使 Python 脚本在 UNIX 上可执行?

你需要做两件事: 脚本文件的模式必须是可执行的 (包括 'x' 位), 并且第一行必须以 `#!` 开始, 后跟 Python 解释程序的路径名。

第一件事可以通过执行 `'chmod +x scriptfile'` 或 `'chmod 755 scriptfile'` 来完成。第二件事可以用许多方法来完成。最简单的编写方法是:

```
#!/usr/local/bin/python
```

作为你的文件的第一行——或诸如此类, 路径名是 Python 解释程序安装在平台上的地方。

如果你愿意让脚本独立于 Python 解释程序而存在, 你可以使用“env”程序。在几乎所有的平台上, 下面的代码行都会工作, 假设 Python 解释程序在用户的 `$PATH` 字典中:

```
#!/usr/bin/env python
```

有时, 用户的环境是非常多, 以致于 `/usr/bin/env` 程序失败; 或者根本没有 `env` 程序。在那种情况下, 你可以尝试下面的程序 (归功于 Alex Rezinsky):

```
#!/bin/sh
"""
exec python $0 ${1+"$@"}
"""
```

缺陷是这样会定义脚本的__doc__字符串。然而，你能通过添加下面的代码行来调整：

```
__doc__ = """...Whatever..."""
```

2.64 如何从列表中删除副本？

一般，如果你没记住重新排序的 List：

```
if List:
    List.sort()
    last = List[-1]
    for i in range(len(List)-2, -1, -1):
        if last==List[i]: del List[i]
        else: last=List[i]
```

如果列表的所有元素都可以用作字典关键字（例如，它们都是可散列化的），通常速度是比较快的。

```
d = { }
for x in List: d[x]=x
List = d.values()
```

同样，对于非常大的列表，你可以考虑二者中更佳的第一个方案。第二种无论什么时候使用都相当好。

2.65 Python 中有众所周知的 2000 年问题吗？

我不知道 Python 1.5 中的 2000 年不足之处。Python 几乎没有日期计算并且对于它所做的事，依赖于 C 库函数。至 1970 年以来，Python 一般把时间说成秒或元组（年、月、日），年由四个阿拉伯数字表示，它不可能生成 Y2K 故障。所以只要你的 C 库是可以的，Python 也应该对。当然，你不能担保你的 Python 代码！

给出免费可用的软件的本质，我必须添加不合法绑定的这条语句。Python 版权布告包含下面的声明：

STICHTING MATHEMATISCH CENTRUM AND CNRI DISCLAIM ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL STICHTING MATHEMATISCH CENTRUM OR CNRI BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNEC-

TION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

这条好新闻是如果你碰到一个问题，你有完整源代码可用于跟踪它并进行调整。

2.66 我想要将方法应用于对象序列的映射版本！请帮助！

要有想象力！

```
def method_map(objects, method, arguments):
    """method_map([a,b], "flog", (1,2)) gives [a.flog(1,2), b.flog(1,2)]"""
    nobjects = len(objects)
    methods = map(getattr, objects, [method]*nobjects)
    return map(apply, methods, [arguments]*nobjects)
```

了解映射、应用、getattr 和 Python 的其他动态功能的秘诀通常是个好主意。

2.67 如何用 Python 产生随机数？

标准库模块“whrandom”实现了随机数发生器。用法很简单：

```
import whrandom
whrandom.random()
```

用 range(0,1) 返回随机浮点数。其他专用发生器也在这个模块中：

randint(a,b) 在范围 [a, b) 中选择一个整数

choice(S) 从给出的序列中选择

uniform(a,b) 在范围 [a, b) 中选择一个浮点数

为了强制随机数发生器初始设置，使用：

```
seed(x, y, z) # 在 1~256 之间选择三个数字作为种子
```

还有一个类 whrandom，你能实例创建独立的若干随机数发生器。模块“random”包含近似不同标准发行版本的函数。在库参考手册中所有这些模块都是文档。注意模块“rand”已经废弃了。

2.68 如何访问串行端口 (RS232)？

有一个 Windows 串行通信模块（用于在 RS232 串行端口上通信）存在于：

<http://www.python.org/ftp/python/contrib/System/siomodule.README>

<http://www.python.org/ftp/python/contrib/System/siomodule.zip>

对于 DOS，尝试 Hans Nowak 的 Python-DX，对它的支持存在于：

<http://www.cuci.nl/~hnowak/>

对于 UNIX，搜索 Deja News（使用 <http://www.python.org./search/>）得到关于作者 Mitch Chapman 写（他的邮件放在这有点太长）的“串行端口”。

2.69 Py15 中 Tk-Buttons 的图像不正常？

它们是正常的，但你现在必须自己保持 image 对象的引用。你必须确信，比方说，全局变量或类属性引用的对象。

从邮件列表清单上引用 Fredrik Lundh：好的，Tk 按钮窗口小部件保持了对内部 photoimage

对象的引用，但是 Tkinter 没有。所以当最后的 Python 引用消失时，Tkinter 告诉 Tk 发布 photoimage。但是因为这个图像正由窗口小部件使用，Tk 不能破坏它，所以不完全。它只是闪烁图像，使它完全透明...是的，在 1.4 版关键字参数处理中有一个缺陷 (bug)，在某些情况下它维持了额外的引用。当 Guido 在 1.5 中修改这个缺陷时，它破坏了好几个 Tkinter 程序...

2.70 math.py (socket.py、regex.py 等) 源文件在哪里？

如果你不能找到模块的源文件，它也许是内置或动态装载的用 C、C++ 或其他编译语言实现的模块。既然这样，你不必有源文件或者它有点像 mathmodule.c，存在于 C 源字典中的某个地方 (不在 Python Path 中)。

Fredrik Lundh (fredrik@pythonware.com) 解释 (在 Python 列表上)：在 Python 中有 (至少) 3 种模块：①用 Python 编写的模块 (.py)；②用 C 编写并动态装载的模块 (.dll, .pyd, .so, .sl 等)；③用 C 编写并和解释程序链接在一起的模块；要得到一系列这些模块，键入：

```
import sys
print sys.builtin_module_names
```

2.71 如何从 Python 脚本中发送邮件？

在 UNIX 上非常简单，使用 sendmail。各系统之间的 sendmail 程序的位置不同：有时，它在 /usr/lib/sendmail，有时在 /usr/sbin/sendmail。sendmail 手册页会帮助你解决难题。这有一些示例代码：

```
SENDMAIL = "/usr/sbin/sendmail" # 发送邮件位置
import os
p = os.popen("%s -t" % SENDMAIL, "w")
p.write("To: cary@ratatosk.org\n")
p.write("Subject: test\n")
p.write('\n') # 划分标题和正文的空白行
p.write("Some text\n")
p.write("some more text\n")
sts = p.close()
if sts != 0:
    print "Sendmail exit status", sts
```

在非 UNIX 系统上 (当然，也可以在 UNIX 系统上)，你能使用 SMTP 发送邮件到最近的邮件服务器。SMTP (smtplib.py) 库包括在 Python 1.5.1 中；在 1.5.2 中才会有文档并进行扩展。这是非常简单的、使用它的交互式邮件发送器：

```
import sys, smtplib
fromaddr = raw_input("From: ")
toaddrs = string.splitfields(raw_input("To: "), ',')
```



```
print "Enter message, end with ^D:"
msg = ''
while 1:
    line = sys.stdin.readline()
    if not line:
        break
    msg = msg + line
# 实际发送邮件
server = smtplib.SMTP('localhost')
server.sendmail(fromaddr, toaddrs, msg)
server.quit()
```

这个方法可以在支持 SMTP 接收器的任何主机上工作；否则，你必须请求为主机的用户。

2.72 在套接字的 connect() 中如何避免阻塞？

一旦它们被连接，select 模块就广泛地为人所知，帮助在套接字上的异步 I/O。然而，如何避免阻塞初始 connect() 调用要少于普通知识。Jeremy Hyton 提出了下面的建议（稍微编辑一下）：

为了阻止 TCP 连接阻塞，可以设置套接字为非阻塞模式。然后，当进行 connect() 时，你就会立即连接或得到包含 errno.errno.EINPROGRESS 的异常。errno.errno.EINPROGRESS 指示连接正在进行中，但是还没有结束。不同的 OS 将返回不同的 errno，所以你必须检查。我能告诉你 Solaris 的不同版本返回不同的 errno。

在 Python 1.5 和近期版本中，可以使用 connect_ex() 来避免创建异常。它只是返回 errno。为了查询，你可以随后再次调用 connect_ex()——0 或 errno.EISCONN 指示你已经被连接——或者你能传递选择的套接字（查看它是否可写）。

2.73 如何指定十六进制和八进制的整数？

为了指定一个八进制整数，用零放在八进制数值之前。例如，设置变量“a”为八进制数值“10”（用十进制是 8），键入：

```
>>> a = 010
```

为了检验这项工作，当在解释程序中时，你能键入“a”并单击 enter，导致 Python 用十进制显示出当前“a”的数值：

```
>>> a
```

```
8
```

十六进制很容易。简单地在十六进制前面放置零，然后是小写或大写字母“x”。十六进制数字可以用小写或大写字母指定。例如，在 Python 解释程序中：

```
>>> a = 0xa5
```

```
>>> a
165
>>> b = 0xB2
>>> b
178
```

2.74 如何每次得到一个单个按键?

这是 UNIX 版的一个答案。有几个解决方案：一些包含 `curses` 的使用，这是要学习的相当大的一块。这也有一些没有 `curses` 的解决方案，归功于 Andrew Kuchling（使用从代码到应用 PGP 风格的随机存储池）：

```
import termios, TERMIOS, sys, os
fd = sys.stdin.fileno()
old = termios.tcgetattr(fd)
new = termios.tcgetattr(fd)
new[3] = new[3] & ~TERMIOS.ICANON & ~TERMIOS.ECHO
new[6][TERMIOS.VMIN] = 1
new[6][TERMIOS.VTIME] = 0
termios.tcsetattr(fd, TERMIOS.TCSANOW, new)
s = '' # 我们将保存字符类型并将它们添加到存储池中
try:
    while 1:
        c = os.read(fd, 1)
        print 'Got character', 'c'
        s = s+c
finally:
    termios.tcsetattr(fd, TERMIOS.TCSAFLUSH, old)
```

为了让任何一种方案工作都需要 `termios` 模块，并且我只在 Linux 上尝试过，虽然它也可以在别处工作。它关掉了 `stdin` 的回应并停用了标准模式，然后从 `stdin` 中每次读取一个字符，注意时间是在每次按键之后。

2.75 如何在 Python 中重载构造函数（方法）？

（这实际上适用于所有的方法，但是不知何故，这个问题通常在构造函数的环境出现。）
你可以用 C++ 编写：

```
class C {
    C() { cout << "No arguments\n"; }
    C(int i) { out << "Argument is " << i << "\n"; }
}
```

在 Python 中, 你必须编写一个使用默认参数捕捉所有情况的单个构造函数。例如:

```
class C:
    def __init__(self, i=None):
if i is None:
    print "No arguments"
else:
    print "Argument is", i
```

这不是完全相等的, 但实际上已足够接近了。你也可以尝试可变长度参数列表, 例如:

```
def __init__(self, *args):
    ...
```

同样的途径可适用于所有的方法定义。

2.76 如何从一个方法到另一个方法传递关键字参数?

使用应用。例如:

```
class Account:
    def __init__(self, **kw):
        self.accountType = kw.get('accountType')
        self.balance = kw.get('balance')
class CheckingAccount(Account):
    def __init__(self, **kw):
        kw['accountType'] = 'checking'
        apply(Account.__init__, (self,), kw)
myAccount = CheckingAccount(balance=100.00)
```

2.77 应该使用什么模块帮助产生 HTML?

查看由 Robin Friedrich 编写的 HTMLgen。它是对应于所有 HTML3.2 标记标签的对象的类库。当你用 Python 编写和希望为产生 Web 页或 CGI 表单等综合处理 HTML 页时它就会被使用。

它能在 python.org 或 Starship 上的 FTP contrib 区域中找到。使用搜索引擎查找最新版本的地点。考虑 DocumentTemplate 也是有用的, 它提供了 Python 代码和 HTML 代码之间清楚的分界线。DocumentTemplate 是 Bobo 对象出版系统的一部分 (<http://www.digicool.com/releases>), 但当然可以单独使用!

2.78 如何从 doc 字符串中创建文档?

使用 gendoc, 由 Daniel Larson 开发的。参见<http://starship.skyport.net/crew/danilo/>。它可以在 Python 源代码中的 doc 字符串中创建 HTML。

2.79 如何读取 (或编写) 二进制数据?

对于复数数据格式, 最好使用 struct 模块。在库参考中有文档。它允许你从包含二进制

数据（通常数字）的文件中，采用字符串读取并把它转换为 Python 对象，并且反之亦然。

例如，下面的代码用 `big-endian` 格式从文件中读取两个 2 字节整数和一个 4 字节整数：

```
import struct

f = open(filename, "rb") # 为可移植性以二进制格式打开文件
s = f.read(8)

x, y, z = struct.unpack(">hhl", s)
```

格式字符串中的 ‘>’ 强制生成 `bin-endian` 数据：字母 ‘h’ 从字符串中读取一个“双字节短整数”（2 个字节），并且 ‘l’ 从字符串中读取一个“四字节整数”（4 个字节）。

因为数据比较正规（例如 `int` 或 `float` 的同类列表），你也能使用 `array` 模块，在库参考中也有文档。

2.80 在 Tkinter 中不能使用键绑定。

最常听到的是，用 `bin()` 方法绑定的事件处理器在响应的按键按下时并没有处理该事件。

最普通的原因是绑定应用的窗口小部件没有“键盘焦点”。检查 Tk 文档得到 `focus` 命令。通常窗口小部件通过单击它给出键盘焦点（但是对标签不适用；参见 `taketocus` 选项）。

2.81 “import crypt（输入 crypt）”失败。

（UNIX）从 Python 1.5 开始，`crypt` 模块就默认停用了。为了启用它，你必须进入 Python 源代码树并编辑文件 `Modules/Setup` 来启用它（删除以 ‘`#crypt`’ 开始的行前面的 ‘#’ 记号），然后重新构建。你可能也要把字符串 ‘`-lcrypt`’ 添加到同一行中。

2.82 有适于 Python 程序的编码标准或样式向导吗？

是的，Guido 已经编写了“Python Style Guide（Python 样式向导）”。参见 <http://www.python.org/doc/essays/styleguide.html>。

2.83 如何冻结 Tkinter 应用程序？

冻结是创建单机应用程序的工具（见 2.28）。当冻结 Tkinter 应用程序时，这个应用程序并不真的是单机的，因为这个应用程序仍然需要 `tcl` 和 `tk` 库。

一个解决方案是用 `tcl` 和 `tk` 库一起发布应用程序，并且在运行时使用 `TCL_LIBRARY` 和 `TK_LIBRARY` 环境变量指向它们。

为了得到真正的单机应用程序，形成库的 Tcl 脚本也必须集成到应用程序中。一个支持工具是 SAM（单机模块），是 Tix 发行版本的一部分（<http://tix.mne.com>）。用激活的 SAM 构建 Tix，在 Python 的 `Modules/tkappinit.c` 内部，执行应用程序对 `Tclsam_init` 等的调用并链接 `libtclsam` 和 `libtkjam`（你可能也包括 Tix 库）。

2.84 如何创建静态类数据和静态类方法？

通过 C 风格的方法

```
self.count=42
```

在 `self` 的字典中创建了一个名为 `count` 的新的、不相关的实例对象，因此覆盖静态类数据名字需要用

`C.count = 314`

无论它是否来自一个方法。

(Tim Peters, tim_one@email.msn.com) 静态数据 (在 C++ 或 Java 语言方面的) 是容易的; 静态方法 (也是 C++ 或 Java 语言方面的) 不被直接支持。

STATIC DATA

例如:

```
class C:
    count = 0 # 调用 C.__init__ 的次数
    def __init__(self):
        C.count = C.count + 1
    def getcount(self):
        return C.count # 或者返回 self.count
```

`c.count` 对于任何像 `isinstance(c, C)` 保持的 `c` 一样也是指 `C.count`, 除非曾被 `c` 本身重载或者在某些从 `c._class_` 回到 `C` 的基类的搜索路径中的类。

STATIC METHODS

在 Python, 静态方法 (与静态数据相反) 是很奇异的, 因为:

`C.getcount`

返回非绑定的方法对象, 没有作为第一个参数应用 `C` 的实例就不能被调用。

得到静态方法的效果的预期方法是通过模块级函数:

```
def getcount():
    return C.count
```

如果你的代码被构造使得每个模块定义一个类 (或者紧密地相关类的层次结构), 这样就会支持想得到的封装。

伪造静态方法的几个曲折的机制是可以通过搜索 DeJaNews 找到的。大多数人感觉这样的治疗比疾病更不好。最不讨厌的方法归功于 Pekka Pessi(<mailto:ppessi@hut.fi>):

```
# 帮助类伪装函数对象
class _static:
    def __init__(self, f):
        self._call_ = f
class C:
    count = 0
    def __init__(self):
        C.count = C.count + 1
    def getcount():
        return C.count
```



```
def run(name, n):
    time.sleep(0.001) # <- - - - - !
    for i in rang(n): print name, i
for i in range(10):
    thread.start_new(run, (i, 100))
time.sleep(10)
```

再多一些暗示:

在结束时代替使用 `time.sleep()` 调用, 使用某种信号机制比较好。一种想法是使用 `Queue` 模块创建队列对象, 让每条线程结束时给队列附加一个标记, 并让主程序从队列中读取和线程一样多的标记。

使用 `threading` 模块代替 `thread` 模块。从 1.5.1 版本以来它就成为 Python 的一部分。它关心所有的这些线程, 并且也有许多其他好的功能!

2.87 关闭 `sys.stdout` (`stdin`、`stderr`) 为什么不能真正关闭它?

Python 文件对象是 C 流顶端抽象的高级层, 依次是 (尤其) 低级 C 文件描述符顶端抽象的中级层。

对于大多数用 Python 创建通过内置 “open” 函数创建的文件对象 `f`, `f.close()` 标志 Python 文件对象从 Python 的观点中关闭, 并且也安排关闭基础 C 流。当 `f` 成为无用信息时, 在 `f` 的析构函数就会自动发生。

但是 `stdin`、`stdout` 和 `stderr` 由 Python 特别对待, 因为它们的特殊状态已用 C 语言给出, 如下操作:

```
sys.stdout.close() # 为 stdin 和 stderr 重复上面的符号
```

标志 Python 级文件对象关闭, 但不关闭相应的 C 流 (提供的 `sys.stdout` 和它的默认值结合在一起, C 流也调用 `stdout`)。

要为这三种情况之一关闭基础的 C 流, 你应该首先确信你真正想要做什么 (例如, 你也许混淆了大量试图完成 I/O 的扩展模块)。如果是这样, 使用 `os.close`:

```
os.close(0) # 关闭 C 的 stdin 流
os.close(1) # 关闭 C 的 stdout 流
os.close(2) # 关闭 C 的 stderr 流
```

2.88 哪种全局数值变化是线程安全的?

(由 Gordon McMillan & GvR 的 `c.l.py` 回答改编的) 在内部使用全局解释程序锁以确信在 Python VM 中每次只有一条线程运行。一般来说, Python 只有在字节码指令之间才能提供线程之间的切换 (它提供切换的频繁程度可以通过 `sys.setcheckinterval` 设置)。每条字节码指令——并且所有的 C 实现代码都通过它达到——因此它们是原子的。

理论上, 这意味着精确的记帐需要对 PVM 字节码实现的精确了解。实际上, 意味着看似原子的内置数据类型 (`ints`、`lists`、`dicts` 等) 的共享 `vrbls` 的操作真的是原子。

例如，这些组合对象中是原子对象（L、L1、L2 是列表，D、D1、D2 是 dict，x、y 是对象，i、j 是 int）：

```
L.append(x)
L1.extend(L2)
x = L[i]
x = L.pop()
L1[i:j] = L2
L.sort()
x = y
x.field = y
D[x] = y
D1.update(D2)
D.keys()
```

这些不是原子对象：

```
i = i+1
L.append(L[-1])
L[i] = L[j]
D[x] = D[x] + 1
```

当它们的引用计数达到零时，代替其他对象的操作也许会调用那些别的对象的 `__del__` 方法，并且能影响这些事情。这对于字典和列表的大规模更新尤其正确。当未确定时，使用互斥体！

2.89 如何适当修改字符串？

字符串是不可改变的，所以你不能直接修改字符串。如果你需要一个具有这种能力的对象，尝试把字符串转换为列表或查看数组模块。

```
>>> s = "Hello, world"
>>> a = list(s)
>>> print a
['H', 'e', 'l', 'l', 'o', ',', ' ', 'w', 'o', 'r', 'l', 'd']
>>> a[7:] = list("there!")
>>> import string
>>> print string.join(a, ' ')
'Hello, there!'
>>> import array
>>> a = array.array('c', s)
```



```
>>> print a
array('c', 'Hello, world')
>>> a[0] = 'y' ; print a
array('c', 'yello world')
>>> a.tostring( )
'yello, world'
```

2.90 我如何从一个函数到另一个函数传递可选参数或关键字参数?

使用 'apply', 如下所示:

```
def f1(a, *b, **c):
    ...
def f2(x, *y, **z):
    -
    z['width'] = '14.3c'
    ...
apply(f1, (a,)+b, c)
```

附录 B 光盘内容

本章要点:

- 运行 CD
- 第一许可证
- 第一用户接口

本书附带的 CD 包含 Python (Linux 和 Windows 版本)、工具 (例如 HTMLgen 和 Grail), Python 常见问题解答 (FAQ) 列表和样本模块及脚本。

B.1 运行 CD

为了使 CD 成为用户友好的并且占用很少的磁盘空间, 安装不是必需的。这意味着传输到你的硬盘上的文件只是你选择拷贝或安装的。

CD 已经被设计为在 Linux 下运行。你可能在图形 HTML 阅读器的任何操作系统上打开和导航 CD, 然而只有一个执行的 Windows 版本。

B.1.1 Linux

- (1) 打开 CD 盘盒并插入 CD。
- (2) 作为 root 用户, 在命令行键入: `mount /dev/cdrom/tmp` (用想要装配 CD 的任何地方代替/tmp)。
- (3) 启动 X 会话 (大多数情况下是 startx)。
- (4) 打开 Netscape Navigator。
- (5) 从 File 菜单中选择 Open, 然后单击 Browse 按钮。
- (6) 浏览 start_here.html。

B.1.2 Windows 95/98/NT4

- (1) 在 CD-ROM 驱动器中插入 CD, 并关闭盘盒。
- (2) 转到控制面板并双击 CD-ROM 驱动器。
- (3) 打开 start_here.html (用 Netscape 和 Internet Explorer 浏览)

B.2 第一许可

你将看到的第一个窗口是第一许可协议。花费一点时间阅读协议，并单击“I Agree”按钮以接受许可，并继续到用户界面。如果你不同意许可，单击“I Disagree”按钮，CD 就不会装载。

B.3 第一个用户界面

Prima 用户界面的打开屏幕包含一个有两个窗格的窗口。左窗格包含磁盘上程序的结构，右窗格显示左窗格中选择的项目的描述页面。

B.3.1 恢复和关闭用户界面

为了恢复窗口，把鼠标放在任意边缘或角上，按住鼠标的左边按钮，拖动边缘或角到一个新的位置。

要关闭和退出用户界面，选择 File、Exit(你的特定的 X 安装程序也许有一个关闭 Netscape 的图形表示)。

B.3.2 使用左边的窗格

如果你想要查看 Tool，单击/Tools。出现一个包含 CD 上不同 Tools 的下拉菜单。同样可以寻找 Script、Module 和 Help。然后简单地单击你想要下载或浏览的工具。它在右边窗口中打开。

B.3.3 使用右窗格

右窗格显示的页面描述了你在左窗格中选择的条目。使用提供的信息来提供你所选择的详细资料——例如可安装程序提供的功能。为了下载这个特别的文件，请按在左窗格中出现的指令。

B.2 第一许可

你将看到的第一个窗口是第一许可协议。花费一点时间阅读协议，并单击“I Agree”按钮以接受许可，并继续到用户界面。如果你不同意许可，单击“I Disagree”按钮，CD 就不会装载。

B.3 第一个用户界面

Prima 用户界面的打开屏幕包含一个有两个窗格的窗口。左窗格包含磁盘上程序的结构，右窗格显示左窗格中选择的项目的描述页面。

B.3.1 恢复和关闭用户界面

为了恢复窗口，把鼠标放在任意边缘或角上，按住鼠标的左边按钮，拖动边缘或角到一个新的位置。

要关闭和退出用户界面，选择 File、Exit(你的特定的 X 安装程序也许有一个关闭 Netscape 的图形表示)。

B.3.2 使用左边的窗格

如果你想要查看 Tool，单击/Tools。出现一个包含 CD 上不同 Tools 的下拉菜单。同样可以寻找 Script、Module 和 Help。然后简单地单击你想要下载或浏览的工具。它在右边窗口中打开。

B.3.3 使用右窗格

右窗格显示的页面描述了你在左窗格中选择的条目。使用提供的信息来提供你所选择的详细资料——例如可安装程序提供的功能。为了下载这个特别的文件，请按在左窗格中出现的指令。

B.2 第一许可

你将看到的第一个窗口是第一许可协议。花费一点时间阅读协议，并单击“I Agree”按钮以接受许可，并继续到用户界面。如果你不同意许可，单击“I Disagree”按钮，CD 就不会装载。

B.3 第一个用户界面

Prima 用户界面的打开屏幕包含一个有两个窗格的窗口。左窗格包含磁盘上程序的结构，右窗格显示左窗格中选择的项目的描述页面。

B.3.1 恢复和关闭用户界面

为了恢复窗口，把鼠标放在任意边缘或角上，按住鼠标的左边按钮，拖动边缘或角到一个新的位置。

要关闭和退出用户界面，选择 File、Exit(你的特定的 X 安装程序也许有一个关闭 Netscape 的图形表示)。

B.3.2 使用左边的窗格

如果你想要查看 Tool，单击/Tools。出现一个包含 CD 上不同 Tools 的下拉菜单。同样可以寻找 Script、Module 和 Help。然后简单地单击你想要下载或浏览的工具。它在右边窗口中打开。

B.3.3 使用右窗格

右窗格显示的页面描述了你在左窗格中选择的条目。使用提供的信息来提供你所选择的详细资料——例如可安装程序提供的功能。为了下载这个特别的文件，请按在左窗格中出现的指令。